[From: Internat.conf. *Methodology & Techniques of Machine Translation*, British Computer Society, Cranfield, 13-15 February 1984]

# Robust Processing in Machine Translation

Doug Arnold and Rod Johnson

Doug Arnold,
Centre for Cognitive Studies
University of Essex.

Rod Johnson
Centre for Computational Linguistics
UMIST.

## Abstract

We attempt to develop a general theory of robust processing for natural language, and especially Machine Translation purposes. That is, a general characterization of methods by which processes can be made resistant to malfunctioning of various kinds.

We distinguish three sources of malfunction: (a) deviant inputs, (b) deviant outputs, and (c) deviant pairings of input and output, and describe the assumptions that guide our discussion (sections 1 and 2). We classify existing approaches to (a)- and (b)-robustness, noting that not only do such approaches fail to provide a solution to (c)-type problems, but that the natural consequence of these solutions is to make (c)-type malfunctions harder to detect (section 3) In the final section (4) we outline possible solutions to (c)-type malfunctions.

## 1. Introduction.

In this paper we attempt to develop a general theory of robust processing and explore its consequences for certain kinds of Machine Translation. Specifically, we assume without argument the goal of a general purpose, fully automatic multilingual MT system to be developed within a highly decentralized organizational framework (for example, the European Commission's EUROTRA project*). The acceptance of such a goal influences our approach in a number of ways.

---

* Our debt to Eurotra is great: collaboration on this paper has developed out of work on Eurotra and has only been possible because of opportunities made available by the Eurotra project. We are also indebted to many colleagues in the project for ideas, insights and support. We would mention in particular Louis Des Tombe, Lieven Jaspaert, Maggie King, Stephen Krauer, Serge Pershke, Mike Rosner, Nino Varile, and our colleagues at Essex and UMIST. The views (and in particular, the errors) expressed in the paper are our own responsibility, however, and should not be interpreted as representing 'official' Eurotra doctrine.

First, the requirement that the system be developed in a highly decentralized organizational framework results in the need for a theory which is both logically strong and highly general, abstracting from many details of purely local relevance.

Second, accepting the goal of multi-lingual MT means that the process of translation cannot be considered simply as a mapping of strings to strings: one is forced to consider the status of intermediate representations of various kinds.

Third, the fact that we consider the issue of robustness at all is a reflection of the difficulty of MT, and the aim of full automation is reflected in our concentration on a theory of robust processing rather than 'developmental robustness'. The general idea of system robustness conflates two quite separate ideas: on the one hand, the idea that systems should be capable of extension and repair by their designers (being, for example, resistant to unforeseen 'ripple effects' under modification). Notice that systems which have only this kind of robustness can never be fully automatic, thus, despite its importance, we will have little to say about this aspect of robustness here. On the other hand, there is the idea that systems should be robust in the sense of capable of dealing with unexpected or deviant data: we will use the terra 'robustness' and related expressions to refer to this.

Three kinds of problem give rise to the need for robustness. For any given process or procedure one may encounter:

case (a). Illegal inputs, i.e. inputs which have not been foreseen. Notice that from the processing (as opposed to the developmental) point of view, it is irrelevant whether the illegality arises from a deficiency in the input itself or a deficiency in the process, i.e., whether it is the input or the process that requires repair.

case (b). Illegal intermediate results. This will occur if some process malfunctions so as to produce deviant output (again the source of this malfunction is irrelevant). It may be that this is not detected until some other process takes this output as input — in which case we have an instance of case (a). However, it may be that in order to produce anything at all some process requires its output to satisfy some condition, so that this is conceptually a separate problem from case (a).

case (c). Suppose both input i and output j of some process are legal objects, it nevertheless does not follow that they have been correctly paired by the process. For example, in the case of a parsing process, i may be some sentence and j some representation. The fact that i and j are legal objects for the parsing process and that j is the output of the parser for input i does not guarantee that j is a 'correct' representation of i. Of course, robust processing should be resistant to this kind of malfunctioning also.

We regard the problem of (c)-type fragility as the most serious, and most resistant to solution: no existing approach is capable of dealing with it, and we will argue that the natural consequence of introducing solutions to (a)- and (b)- type errors is a proliferation of the more dangerous and insidious (c)—type errors.

In this paper we briefly review existing approaches to process robustness in Natural language processing and MT, with some discussion of their deficiencies in relation to our general goals, and the goal of (c)-type robustness in particular, and attempt to develop a partial solution to the problem of (c)-type robustness.

## 2. Basic notions and background assumptions.

To begin with it is useful to narrow the discussion somewhat by noting a number of ways in which the problem of system fragility in the kind of environment we are concerned with differs from that encountered in Natural Language Processing (NLP) generally. For example, we are not particularly concerned with failures that result from different kinds of mis-spelling, and mis-segmentation in the input, since we imagine that texts submitted for translation will already have been processed for such errors (perhaps automatically), nor are we concerned with dialogue related problems such as highly fragmentary input, interjections, or false starts, though these clearly present serious difficulties in some kinds of NLP (e.g. in front ends to Expert Systems).

On the other hand, certain common solutions to problems of fragility are obviously not open to us. In particular, the aim of a general theory for robust MT excludes the use of highly domain specific knowledge such as is exploited by many special purpose NLP systems (cf. Hayes and Mouradian [5]), and the fact that we are concerned with translation militates against the disregard for input that is characteristic of some robust systems (it is displayed in an extreme form in e.g. PARRY [2]): it is not enough that an MT system behaves robustly, producing some output, it should produce output that stands as nearly as possible in the 'translation of relation to its inputs. Finally, though we are not discussing the issue of developmental robustness here, we will obviously prefer robust processing to preserve a high degree of transparency — we take it as axiomatic that general purpose MT systems must be capable of extension and repair.

From the point of view we adopt, it is possible to regard an MT system as a set of processes implementing relations between representations (input and output texts can be considered representations of themselves). We distinguish three different kinds of relation:

(1) The correct, or intended relation R between

representations. E.g. the relation 'is a (correct) translation of, which pairs texts in one language with texts in another. We have only pre-theoretical and rather vague ideas about Rs, in virtue of being bi-lingual speakers, or having some intuitive grasp of the semantics of artificial representations.

(2) A theoretical construct T that is supposed to embody R.

(3) A process P that is supposed to implement T.

The need to distinguish R from T and P is obvious: it is possible to perform evaluations of a system only by comparing actual performance of P against ideas about R. However, it is also necessary to distinguish T from P. In some cases T may exist as a separate entity (e.g. if P is a process that implements an explicit grammar of some sort) so that the need to separate evaluation of T (the grammar) and P (the programs) is obvious. However, even when T does not exist as an explicit set of propositions, it is useful to consider it separately, as an interface between pre-theory and process. It is a fact that every actual process implements a theory of inputs and outputs, however implicit, and the existence of such an interface is essential to our presentation. We want to distinguish carefully between evaluation of T (e.g. checking the adequacy of ones representational devices) and evaluation of P (checking how far an implementation delivers representation). We are concerned with automatic approaches to error detection and repair, and we can imagine no automatic method for checking the correctness of a representational device (T). For this reason we want to ignore questions of how far T can be considered a good instantiation of R.

Thus, in what follows, we will simply assume that T is both well-defined and a correct embodiment of R (e.g. in the case where T is a theory of translation between L and L', this assumption says that the membership of L, and L' is well-defined and members of L are paired with their correct translations in L'). Realistically, in the context of NLP, the assumption of the correctness of T in relation to R will amount to assuming it to be the most correct available — that all T' distinct from T are in some way less correct — in fact, this assumption is sufficient for our purposes.

It will considerably simplify the exposition below if T can be regarded as a function; this can be achieved if we abstract away from the phenomenon of ambiguity (it will not matter if we regard T as a relation between individual representations, or between individual representations and sets of representations which are equivalent). While we think this simplification is essential to the exposition here, it is regrettable, since the inter-relation of ambiguity and robustness is an important matter.

Finally, we will assume that hardware and low-level software operate error-free and that P can be guaranteed to terminate for

all inputs: there are well known ways to ensure system robustness at this kind of level (e.g. termination is guaranteed by simply restricting allocation of resources to P), so that this assumption seems unproblematic.

Given these assumptions the possible sources of error in *a* system or process P are restricted to two:

> Problem 1: Correctness of P. P is not a correct implementation of T. One might expect this situation in cases where T is extremely complex, which we consider will be a common situation in NLP and MT — even for domains which are reasonably well understood, theories are extremely complex, and there are severe problems devising implementations of them.
>
> Problem 2:_ Completeness of_ T. T, while correct, is not complete. We have assumed that T is correct, i.e. that it correctly pairs all items in its domain with items in its range. It is not a consequence of this assumption that the domain of T is co-extensive with the set of actual inputs to P. In fact, in realistic NLP we expect the set of actual inputs regularly to be a strict superset of the domain of T, for non-trivial Ts. Even if T were to include what amounts to *a* complete grammatical description of the input language this would be so, since we can expect some inputs of only marginal grammaticality, and all languages allow scope for creativity that is under determined by the rule system (e.g. creation of new, derivationally simple terms).

In principle, it might turn out that a combination of advances in understanding together with restrictions on input might eliminate both sources of error. It seems reasonable to disregard this possibility, and assume that robust processing will always be necessary.

We can now state the possible manifestations of system fragility described in the introduction more concisely:

> case (a): $P(x)=\emptyset$. i.e. P halts producing Ø output for input x. This is the effect of illegal (unforeseen) input.
>
> case (b): $P(x)=z$ where z is not a legal output for P according to T.
>
> case (c): $P(x)=y$, where y is a legal output for P according to T, but is not the intended output according to T. i.e. y is in the range of T, but $y \neq T(x)$.

We should also be precise about the alternative to malfunctioning: 'correct' processing, and in particular processing which avoids (c)-type malfunctions. We will speak of outputs being 'T-correct' when they are the results of such processing. By abstracting away from ambiguity, we are able to consider T to be a function. It does not follow that the inverse

of T (that is, $T^{-1)}$ is also a function, since T may be many-to-one. This complicates the definition of T-correctness slightly:

> Given P(x) = y, and a set W such that for all w in W, T(w) = y, then y is T-correct with respect to w iff x is a member of W.

When x is not a member of W, there is a (c)-type error.

## 3. Existing approaches.

We now present a classification of existing solutions to instances of (a)- and (b)- type fragility. None of these provides a solution to (c)-type fragility.

### Case-(a) errors: 'input' robustness:

Case (a) errors, where P halts without producing any output for input x, have their source in a mismatch between the expectations of P and the data it is presented with. They are the most commonly considered in the literature. There are two basic approaches to making systems 'input robust':

> (i) to call some alternative process P' to manipulate the data so that it satisfies the expectations of P: for example, the LIFER [6] approach to elliptical input involves attempting to restore the ellipsis, so that the input can be processed by the normal rules. Notice that P' cannot guarantee to do more than make x formally acceptable to P, which will generally lead to (b)- and (c)- type problems, as later processes find inconsistent or incomplete information. The alternative would be fortuitous, and not very likely: (1) that P and P' together simply constitute a more correct implementation of T, hence solving problem 1; (2) that P and P' together implement a theory T' which differs from T only in having a larger domain, hence solving problem 2.
>
> Thus, though if it is successful, this strategy will eliminate case (a) errors, case (b) and (c) errors will remain, and are likely to be more wide-spread.
>
> (ii) Provide some mechanism for modifying the expectations of P: e.g. by calling some alternative process P' which embodies weaker expectations about data, or by attempting some temporary re-arrangement of P (cf. Kwasny and Sondheimer [7]), or simply relaxing P's requirements on inputs (as in 'Preference' type approaches, (Wilks [9])). Variants of this approach are extremely common: (cf. Weischedel and Black [8], HEARSAY [4], and in general, systems that favour a bottom up approach to exception processing). The effect of this is to create some new process P' which accepts a superset of the inputs of P. Again, it is unlikely that this will simply yield a more correct implementation of T. It is more likely that P'

implements a new theoretical construct T', distinct from T, and with a wider domain.

If this is successful, it will eliminate type (a)-errors, but now the existence of type-b and c errors is likely: given that T and T' are different, they may well differ in terms of range, as well as domain, and anything which P' delivers outside the range of T constitutes a (b)- or (c)-type error.

We return to the case where the ranges of T and T' are known to coincide in section 4.

(Notice, incidentally, that if making P robust does involve implementing an alternative to T, then the assumption about the correctness of the theory in relation to R is no longer valid, so that even if our processor is guaranteed to deliver what the theory T' predicts it should, there is no guarantee that this is what is really intended (i.e. what is correct for R). Of course, it might, in principle turn out that T' actually better approximates to R, and performs better by accident — such accidents are extremely improbable, and we think we can disregard them).

Case-(b) errors; 'output robustness'

Case-(b) errors, where the output of P is ill-formed according to T can be trapped straightforwardly, by imposing a 'goal filter' or well-formedness check on the output of P (as in TAUM AVIATION [1], where the output of Transfer is checked in this way). This approach is particularly useful where it is expected that collectively coherent and useful sub-parts of the output of P can be salvaged by the filter.

The effect of this is that P either produces 0 (an (a)-type error), or something more well-formed, so that the likely result of filtering output in this way is to produce a proliferation of type-a errors as P is unable to produce any output that satisfies its goal. There are a number of ways to avoid this, the value of which is that processing successfully performed by P is not wasted, as it would be if P simply failed, producing Ø. The obvious danger of all this is that the 'fall back' output of P may be illegal or unusable for some process that P feeds.

We can distinguish three methods for achieving 'output robustness':

(i) introduce a fall-back process that will massage the output so that it becomes well-formed according to the goal. This is suitable when the actual output is very close to the desired output (e.g. if P is to output a complete labelled tree, the case where the actual output lacks only one label could be saved by a process which introduces a 'wild card label' that matches on anything,

and thus satisfies the goal).

(ii) introduce some 'ranking' of successively weaker conditions in the output, so that if output fails one, it may still be passed by another less stringent filter. This would be a natural part of a strategy implementing a version of 'Preference' [9] to make sure that if a process produces a number of inputs, the best of these is output, even if it is less than perfect.

(iii) It may be that P fails to produce an output of the desired kind, but that the system that includes P has been set up so that it has some alternative strategies which it is able to employ, using intermediate results of P. The example we have in mind is of a Transfer based MT system (such as EUROTRA) in which analysis aims to produce *a* semantic representation, as input to normal transfer, but which includes a 'safety net' transfer module employing the syntactic representation that analysis routinely builds and maintains as it is attempting to produce the semantic representation. Here the possibility of subsequent procedures failing by being unable to utilize the 'fall-back' output of P is extreme, and this solution incurs a considerable developmental overhead, as alternative processors must be designed to cope with the fall-back output.

Of course, a fourth alternative is to simply allow P to produce Ø, and rely on standard (a)-type solutions. Notice that in any case, trapping (b)-type errors is likely to lead to some (a)-type errors. Though the combination of P and P' may accept a superset of the inputs of P, the well-formedness check will mean that P itself sometimes outputs Ø, and it is likely that P and P' together will sometimes produce imperfect output that will cause some later process to fail, producing Ø.

Moreover, making P more robust by weakening the well-formedness check on the lines of (i)-(iii) has the same sort of pernicious effect as (a)-type solutions. Again, it is unlikely that P and P' together are simply more correct implementations of T, or that they implement T' which differs from T only in having a wider domain. The most probable effect of making P output robust is that it now implements a version of a theoretical construct T' which differs from T both in domain (since output robust P may well accept a superset of P) and range (since robust P is likely to produce a superset of P).

The problem of (c)-type errors is now acute: the effect of increasing robustness has been to ensure that some approximation to superficially correct output is reliably produced. But, of course, there are many cases where no output is to be preferred to one which is superficially well-formed, but actually wrong as a representation of the input.

In fact, the situation is somewhat worse, for not only has the

number of (c)-type errors increased as processing has been made (a)- and (b)- robust, but introducing (a)- and (b)- robustness has weakened our grip on the notion of correctness in relation to R itself, since the modifications P has undergone in being made robust have meant that it no longer implements T, but T', which may be distinct from, and weaker than T.

From this we can draw an immediate and obvious conclusion about the need to distinguish sharply between 'ideal' and 'robust' processing. We have assumed that T is a correct (or the best approximation to a correct) instantiation of R, so that there is simply no point in checking for errors in relation to anything other than T (such a check would have no clear relation to the intuitive ideas about correctness that constitute R). If it is to be worthwhile, then, checking for (c)-type errors requires that we are able to distinguish T from the T' which is implemented by a robust version of P. Theoretically, this is unproblematic. However, in a domain such as MT it will be rather unusually for T and T' to exist separately from P and P' that instantiate them. Thus, the need to separate 'ideal' and 'robust' processing in this context comes down to the need to be able to separate out those aspects of a robust processor that implement 'ideal' T. This will normally mean distinguishing sharply between P and P'. This is worth pointing out, since this distinction is not one that is made in most robust systems.

In the final section we discuss some ways in which automatic evaluation of P might be made feasible, and (c)-type errors detected.

## 4. Two approaches to (c)-type robustness

There are two distinct issues with respect to (c)-type errors: detection and repair. Clearly, the second presupposes the first, and though one of the approaches we describe yields a method repair, directly, we will have relatively little to say at this time about repair as such.

(c)-type errors differ from (a)- and (b)-type errors in an important way: (a)- and (b)-type errors can be detected quite simply by a check on well-formedness with respect to the domain and range of T respectively. (c)-type errors can only be detected with certainty by computing the pairing of elements of the domain and range of T. This is, of course, the task which P itself was designed to do.

What is required is an implementation of a relation that pairs items in exactly the same way as T: the obvious candidate is the inverse of T, that is, $T^{-1}$. We will return to this below, but notice this will only be feasible provided there is known to be a way of implementing $T^{-1}$ which is considered to be reliable.

However, we might consider *a* partial solution derived from a well-known technique in systems theory: insuring against the

effect of faulty components in crucial parts of a system by computing the result for a given input by a number of different routes. For our purposes, the method would consist essentially in implementing the same computation in parallel a number of times and using statistical criteria to determine the correctness of the computation. We will call this the 'statistical solution'. (Notice that certain kinds of system architecture make this quite feasible, even given real time constraints.) Clearly, however, while this should significantly improve the chances that output will be correct, it can provide no guarantee.

Moreover, the kind of situation we are considering is more complex than that arising given failure of relatively simple pieces of hardware. This is because we have to consider three distinct cases of failure.

> (1) Where we believe T to be adequate, but expect P will be error prone. (c.f. Problem 1: Correctness of P) In this case the obvious solution would involve implementing T many times, as independent Ps, taking the result that is most frequent.

The other two cases arise where we are relatively confident of the implementation, but are concerned with the incompleteness of T (cf Problem 2: Completeness of T). On the face of it, we can proceed in this case by implementing a number of different T's. However, this leads to new problems, as we have already suggested above.

> (2) If the range of all these Ts coincides, then statistically at least, this method should yield adequate results. Normally, however, it is likely to be difficult to construct distinct T's which have this property, and which at the same time correctly embody R. Nevertheless, this would appear to be a natural way of extending approaches to (a)-type robustness to cope with (c)-type errors.

> (3) We expect the normal situation to be that the ranges of the different T's are distinct. The problem now is that we have no basis for comparison of the results, and hence no longer any sensible statistical criterion. Notice that the technique of producing robustness in response to (a) and (b) errors virtually guarantee that this situation arises. (Note also that the apparent solution offered by some systems (e.g. GETA [3]), ranking Ts and accepting the results that conform to the highest valued T subject to some measure of completeness, is evidently not a solution to the problem here, since it offers no guarantee that the output is T-correct with respect to any T). Thus in this case, if we wish to check for (c)-type errors we have no alternative but to implement a process which computes the inverse of T.

The statistical solution is attractive because it shifts the emphasis in coping with (c)-type errors from detection to repair,

effect of faulty components in crucial parts of a system by computing the result for a given input by a number of different routes. For our purposes, the method would consist essentially in implementing the same computation in parallel a number of times and using statistical criteria to determine the correctness of the computation. We will call this the 'statistical solution'. (Notice that certain kinds of system architecture make this quite feasible, even given real time constraints.) Clearly, however, while this should significantly improve the chances that output will be correct, it can provide no guarantee.

Moreover, the kind of situation we are considering is more complex than that arising given failure of relatively simple pieces of hardware. This is because we have to consider three distinct cases of failure.

> (1) Where we believe T to be adequate, but expect P will be error prone. (c.f. Problem 1: Correctness of P) In this case the obvious solution would involve implementing T many times, as independent Ps, taking the result that is most frequent.

The other two cases arise where we are relatively confident of the implementation, but are concerned with the incompleteness of T (cf Problem 2: Completeness of T). On the face of it, we can proceed in this case by implementing a number of different T's. However, this leads to new problems, as we have already suggested above.

> (2) If the range of all these Ts coincides, then statistically at least, this method should yield adequate results. Normally, however, it is likely to be difficult to construct distinct T's which have this property, and which at the same time correctly embody R. Nevertheless, this would appear to be a natural way of extending approaches to (a)-type robustness to cope with (c)-type errors.
>
> (3) We expect the normal situation to be that the ranges of the different T's are distinct. The problem now is that we have no basis for comparison of the results, and hence no longer any sensible statistical criterion. Notice that the technique of producing robustness in response to (a) and (b) errors virtually guarantee that this situation arises. (Note also that the apparent solution offered by some systems (e.g. GETA [3]), ranking Ts and accepting the results that conform to the highest valued T subject to some measure of completeness, is evidently not a solution to the problem here, since it offers no guarantee that the output is T-correct with respect to to any T). Thus in this case, if we wish to check for (c)-type errors we have no alternative but to implement a process which computes the inverse of T.

The statistical solution is attractive because it shifts the emphasis in coping with (c)-type errors from detection to repair,

and because they avoid the need to map backwards from output to input. Such solutions are certainly worth further consideration. However, realistically, we expect the normal situation to be as described in (3), so that it is worthwhile to consider the feasibility 'inverse solutions' involving construction of $P^{-1}$ implementing the inverse of T.

The basic method here would be to compute an enumeration of the set of all possible inputs W that could have yielded the actual output, given T, and some hypothetical ideal P which correctly implements it. (Again, this is not unrealistic; certain system architectures would allow forward computation to proceed while this inverse processing is carried out).

To make this worthwhile involves two assumptions:

1. That $P^{-1}$ terminates in reasonable time. This cannot be guaranteed, but it can be rendered a more reasonable assumption by observing characteristics of the input, and thus restricting W (e.g. restricting the members of W in relation to the length of the input guarantees that W is finite, and for some Ts it may be possible to exploit more interesting characteristics of internal structure).

2. That construction of $P^{-1}$ is somehow more straightforward than construction of P, so that $P^{-1}$ is likely to be more reliable than P. In fact this is not implausible for some applications (e.g. consider the case where P is a parser: it is a widely held idea that generators are easier to build than parsers).

Granted these assumptions, one simply examines the enumeration for the input if it is present. If it is present, then given that $P^{-1}$ is likely to be more reliable than P, then it is likely that the output of P was T-correct, and hence did not constitute a (c)-type error. At least, the chances of the output of P being correct have been increased.

In the nature of things, we will ultimately be lead to the original problems of robustness, but now in connection with P-1. For this reason we cannot foresee any complete solution to problems of robustness generally. What we have seen is that solutions to one sort of fragility are normally only partly successful, leading to error of another kind elsewhere. Clearly, what we have to hope is that each attempt to eliminate a source of error nevertheless leads to *a* net decrease in the overall number of errors.

REFERENCES

1. BOURBEAU, L. (1981): Linguistic documentation of the computerized chain of the TAUM-AVIATION system. TAUM, University of Montreal.

2. COLBY, K. (1975): Artificial Paranoia Pergamon Press, Oxford.

3. BOITET, CH., & NEDOBEJKINE, N. (1980) 'Russian-French at GETA: an outline of method and a detailed example' RR 219, GETA, Grenoble.

4. ERMAN, L.D. and LESSER, V.R. (1978) 'HEARSAY-II: Tutorial Introduction and Retrospective View', Tech. Rep. Computer Science Dept, Carnegie Mellon University.

5. HAYES, P.J. and MOURADIAN, G.V. (1981): "Flexible parsing", AJCL 7, 4:232-242.

6. HENDRIX, G.G. (1977): "Human Engineering for Applied Natural Language Processing". Proc 5th IJCAI, 183-191, MIT Press.

7. KWASNY, S.C. and SONDHEIMER, N.K. (1981): "Relaxation Techniques for Parsing Grammatically Ill-formed Input in Natural Language Understanding Systems". AJCL 7, 2:99-108.

8. WEISCHEDEL , R.M, and BLACK, J. (1980) 'Responding Intelligently to Unparsable Inputs' AJCL 6.2: 97-109.

9. WILKS, Y. (1975): "A Preferential Pattern Matching Semantics for Natural Language". A.I. 6:53-74.