Error Detection, Recovery, and Repair in the Translation of Inorganic Nomenclatures. 1. A Study of the Problem

I. Luque Ruiz,*,‡ J. L. Cruz Soto,† and M. A. Gómez-Nieto‡

E.U.P., Avda, Menéndez Pidal s/n, and Science Faculty, Avda, S. Alberto Magno s/n, Córdoba University, 14071-Córdoba, Spain

Received June 13, 1995⊗

The treatment of errors generated in the transmission of chemical information from humans to machines is examined. Every type of human-machine communication requires a translation subprocess to deal with the various possible representations of knowledge; the present study was designed to consider errors occurring in such a subsystem. Regardless of the model employed, translation of knowledge is performed over several successive stages; at each stage different types of errors may be detected. A method is proposed to classify these errors according to the stage at which they occur, thus facilitating the generalization of the process. An analysis of errors occurring during the translation of inorganic chemical names is also presented. The particular grammatical features of different nomenclatures would require a study of the errors appearing in the human-machine communication process in modern chemistry-oriented systems, a problem which has only been touched upon in the systems proposed to date. Attention here is centered on lexicographic errors, since that stage of the translation process is completely independent of the model employed and might thus be useful in a more general sense.

1. INTRODUCTION

In the development of computer systems for use in the various areas of arts and sciences, the problem of human-to-machine communication persists. Current expectations of such systems involve the recognition of *natural* or *scientific* language in order to enable the user to represent his/her knowledge of a problem. The system must recognize the sentences or phrases of this language and translate them to an abstract representation which may then be processed by the computer. This switching between different representations is translation usually via a compilation.

Independently of the model on which it is based, a translator is a program that analyzes, over successive stages, sentences of a source language and builds sentences in a target language. These stages involve orthographic, lexicographic, syntactic, and semantic analysis of the input, checking each time that sentences comply with the rules of the input language before finally translating them to sentences that are grammatically correct in the target language (see Figure 1).

In the development of expert systems, computer-assisted learning systems, tutorial systems, etc., this translation process is always required. The computer system has to be equipped with a translator capable of recognizing the sentences by which a user represents knowledge and of translating these into a new representation close to machine language; once this process has been carried out, the machine must then be able to express its internal information in a form that the user will understand.

In recent years, a great deal of research has been performed in the area of translation of chemical knowledge to and from different formats. Surprisingly, though, little attention has been paid to one problem arising during the process—that of error handling.[1−5]

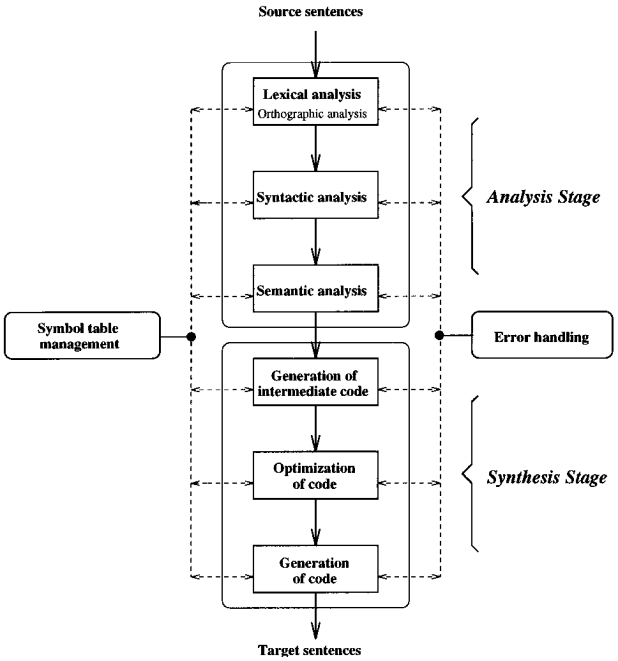

Figure 1. Stages in the process of knowledge translation.

Due to the nature of human−machine interaction, a translator or compiler must more often deal with incorrect source language sentences than correct ones.[6] For this reason the translator should be equipped with a strategy to handle incorrect input, based on previously-defined grammar rules. Faced with an input of erroneous source language sentences, a translation system may react in one of the following ways, according to the classification proposed by Horning:[7]

1. **Unacceptable responses**

(a) *Incorrect responses (error not reported)*

i. The translation process aborts.

† Science Faculty. e-mail: jlcruz@sun630.uco.es.

‡ E.U.P. e-mail: malgonim@lucano.uco.es :: mallurui@sun630.uco.es.

⊗ Abstract published in *Advance ACS Abstracts,* January 1, 1996.

0095-2338/96/1636-0007$12.00/0

ii. The translation process falls into an endless loop.
iii. The translator continues producing wrong target language sentences.

(b) *Correct responses (but hardly practical)*

i. The translator reports the first error found and closes down the translation process cleanly.

2. **Acceptable responses**

(a) *Possible responses*

i. The translator reports the error and reestablishes the process, searching for any further errors.
ii. The translator reports the error, repairs it, and carries on with the translation, producing valid target language sentences.

(b) *Responses impossible with current techniques*

i. The translator corrects the error and produces the target language sentences which the user (or programmer) had intended to write.

The lowest level of response is where the translator has no possibility of replying to errors. In such a case the system would partially or completely lose control, possibly bringing a sudden halt to the process and lapsing into an infinite loop and/or producing incorrect target language sentences. At the next level up, the developer will have taken the possibility of errors into account, reporting these as they appear and halting the translation process. A translation system which only produced this level of response would not be acceptable in most cases.[8]

When faced with errors, the lowest level of response required of a system is where the translator as well as reporting the error not only recovers but also carries on searching for and reporting any other errors in the source language. The next higher acceptable level is where in addition to detecting, reporting, and recovering from an error, the translator is able to repair it, i.e., alter the input sentence containing the error substring and build a new, correct one. The ideal response level would be, however, one in which the system not only detected, reported, recovered from, and repaired the error but also produced a repaired sentence that was exactly what the user had intended to generate; this would, of course, presuppose a translator which knew exactly what the user was thinking—somewhat beyond the limitations of current systems.[9]

The behavior of the translator when dealing with errors could be seen as a particularly valid measurement of its quality. A desirable system would incorporate an acceptable subsystem to take care of errors; one that did not lead to greatly-increased processing times, but this is difficult to accomplish. Thus, for any viable system it is necessary to decide how much increase is tolerable.

The process of translation from one representation of knowledge to another is generally performed as shown in Figure 1. Firstly, the lexical analyzer or *scanner* analyzes the sentences from the source language, ensuring that they are composed of morphemes belonging to that language's vocabulary and generating a new string formed by tokens or valid grammatical structures. Next, the syntactic analyzer or *parser* analyzes this string of tokens, checking for correct order (the structure of the input sentence must satisfy the grammatical rules defining the source language). In the next stage, usually performed in parallel to the previous one, the semantic analyzer checks that morpheme values are valid for the language. The final stages involve the synthesis of a new string in the target language, one which should perfectly represent the knowledge contained in the input string. Each of these processes is dependent on a set of symbol tables containing definitions of the components and basic structures of the source language, and any errors occurring during translation are managed ad hoc by dedicated procedures.[10−12]

Thus, the detection of errors arising during these processes is carried out by different subsystems, depending on the nature of the error. Errors may be classified according to the point in the translation process at which they are detected, as follows:

Orthographic: errors caused by the presence in the input string of illegal characters or by the excess or absence of morpheme delimiters. Take, for example, the string *#sulfuric_%_ _ _acid*, which contains the illegal characters *(#) and (%)*, in addition to extra spaces (the underline character *(_)* is used here to represent spaces).

Lexicographic: errors caused by the presence in the input string of substrings that fail to coincide with any of the morphemes defined by the grammer; in other words, vocabulary which does not appear in the language. For instance, the string *sodium sulate*, in which no morpheme coinciding with the substring *sul* appears in the grammar.

Although orthographic and lexicographic errors could be considered to be of one single type, as they might both be attributable to typing mistakes, it is convenient to class them by two distinct categories or error groups since they are detected at different stages in the translation process, namely during orthographic and lexicographic analysis.

Syntactic: errors caused by the presence in the input string of a construction or aggregation of morphemes which violates the rules of the defined grammar. For example, the string *sodpotassium oxide* which contains two consecutive root-symbol morphemes, i.e., the morphemes *sod* and *potass*.

Semantic: errors produced by the presence in the input string of a grammatical structure which, while correct, has no real meaning. By way of example, take the string *cobaltate oxide*, which is correct both lexically and syntactically, but not semantically, since this suffix morpheme may only have the values *ic* or *ous* when accompanied by a root-symbol morpheme and when the Keyword morpheme *oxide* is present.

Construction of target-language sentences: errors that occur when there is a clash between the grammar-defined semantic rules and the values given to recognized morphemes, leading to failure of the target-language sentence-construction functions.

System: Errors due to system breakdown, software "bugs", unforeseen situations, hardware faults, etc.

Any translator system should be prepared for the appearance of errors of whatever kind and be ready to control the "damage" these may produce. A translator may "fail", but only under controlled conditions.

1.1. Lexical Errors. Lexical errors are detected at the lexicographic analysis stage. Few errors are in fact detected at this point since the lexicographic analyzer has a highly restricted vision of the language of the grammar to which the input sentences must belong. Thus, the scanner only deals with defined terminal symbols or with the vocabulary of the language as defined by lexemes and their associated actions. Definitions are made by formal propositions of the type (where the | symbol represents optionality):

morpheme | *morpheme* | ... | *morpheme* : lexical action

Lexical errors are produced by substrings belonging to input sentences for which the lexical analyzer can find no recognizable predefined pattern. These incorrect substrings may be attributable to (a) incomplete knowledge of the source language, (b) typing mistakes, or (c) spelling mistakes.[9,13]

In any computer system, typing and spelling mistakes will be the most frequent source of lexical errors. Shaffer and Hardwich[14] reported that, in typing, the substitution of one character by another was the commonest error, followed by the omission and insertion of a character. Bourne[15] examined typographical errors in a large number of bibliographical databases and found an average of 11% mistakes in lexemes. Damerau[16] reported that 80% of lexical errors in programming language source code were due to omission, insertion, substitution, and transposition of adjacent characters in lexemes. Morgan[17] stated that a great number of errors were attributable to ignorance of morpheme structure—spelling mistakes—and not to mistakes in typing.

Since the above-mentioned research was published, a great deal of effort has been directed at analyzing the sources and studying the features of typographical errors; generally speaking, this research has been wholly or partly aimed at development of tutorial systems, computer-assisted learning, study of both natural and programming languages, recognition of patterns in text, images, and, particularly, speech, access to large databases, lexicon management, etc.[13,18−26]

During the lexical analysis process, if the lexicographic analyzer comes up against a string which it does not recognize as one of the morphemes or words defined in the language's vocabulary, it may react in several very different ways:

- If the error is due to the presence of illegal characters, i.e., characters not defined in the language's alphabet, these may be eliminated and another attempt is made, this time to recognize the newly-generated string.
- If the string is still unrecognizable, the analyzer may do the following:
  - —Ignore the string and carry on reading the input until it finds a string which corresponds to one of the defined morphemes. This type of action would generally result in the error being hidden, to be discovered later by the syntactical analyzer. The lexical analyzer may report the error and leave the syntactical analyzer to perform its analysis and search for other kinds of errors. While this approach may be acceptable in a programming environment, it could lead to the syntactical analyzer reporting errors that had been produced by the elimination of the offending lexeme detected by the lexical analyzer and occasionally result in an avalanche of self-generated errors.
  - —Correct the error string by deleting, inserting, replacing, or transposing characters so that the new string corresponds to one of the morphemes defined in the grammar. In the simplest case, the error would be one of the type described by Morgan where it could be taken for granted that there had only been one change to the intended string. Obviously, this is not always the case.

A translator system would thus set about repairing lexical errors by finding one or more morphemes to successfully repair the error string.

This clearly involves *matching*, where given a pattern p of length m, a text t of length n is searched for all occurrences of the given pattern. Indeed, much work has been dedicated to the matching of strings. It is a subproblem in many fields such as text editing, symbol manipulation, and data recovery, giving rise to the invention of terms like "stringology" to describe this area of research.[27−31]

Some interesting algorithms for exact string matching have been proposed by Knuth−Morris−Pratt (KMP) and Boyer−Moore (BM); these are detailed in the bibliography.[32−38] With respect to the repair of lexicographic errors, however, there exists a different state of affairs. Here we must find a pattern (one of the defined morphemes) that is *like*, *approximately like*, or *most like* the error string; that is to say, we have decide if two strings may be described as *similar*, a process known as *approximate string matching*.[39] Hall[40] has proposed that for the calculation of similarity the difference function should be used, defined thus d: $S \times S \rightarrow R$, with the following properties:

$$d(p,t) \geq 0$$

$$d(p,t) = 0, \quad \text{only if } p = t$$

$$d(p,t) = d(t,p)$$

$$d(p,t) + d(t,r) \geq d(p,r) \text{ (triangular inequality)}$$

This metric is based on the concept of *edit difference* or *distance*, defined as the minimum number of operations necessary for two strings to be identical.[41] Its formal definition is thus as follows.

Let $\sum$ be a finite alphabet. An edit operation involves a pair of strings $(a,b) \neq (\lambda,\lambda)$ on $\sum$ of length zero or one. A string x is said to be the result of another string w and is represented as $x \rightarrow w$, if the strings α and β exist, such that $x = \alpha b \beta$ and $w = \alpha a \beta$. We can say that (a,b) is a substitution if $a \neq \lambda$ and $b \neq \lambda$, an erasure if $a \neq \lambda$ and $b = \lambda$, and an insertion if $a = \lambda$ and $b \neq \lambda$. An edit distance of k is given by the set k of (a,b) operations required in order to make two initially different strings x and w identical.

In this way, the problem of correcting a lexical error may be formulated as the problem of finding all the morphemes whose difference d(*morpheme*, *error*) is below a predetermined threshold k, or else of finding the morpheme with the lowest d(*morpheme*, *error*) value.

Wagner and Fischer[42] reformulated the problem basing their work on an optimization model. They iteratively calculated the value of the function $f(i,j)$, where i and j determined the ith and jth characters corresponding to the error string (t) and the pattern (p), respectively, and where

$$f(0,0) = 0$$

$$f(i,j) = min[f(i-1, j) + 1, f(i,j-1) + 1, \qquad (1)$$

$$f(i-1, j-1) + d(t_i,p_i)]$$

$$where: d(t_i,j_i) = 0 \; if \; t_i = p_j$$

$$d(t_i,p_j) = 1 \; if \; t_i \neq p_j$$

These methods may be viewed as a problem of finding the shortest path down a tree or, when weighting is employed, the lowest-cost path, and several algorithms have been

proposed to this end.[43] Other authors have put forward linear and parallel algorithms, adaptations of the Boyer−Moore approximate matching algorithm based on complex structures for the representation of strings such as suffix arrays, suffix-prefix trees, adaptations of algorithms for document, ideogram, and voice recognition (e.g., developing systems to recognize dyslexia), the application of element (object) recovery for the construction of software and images, and the application of mathematical and statistical models, to mention just a few.[44−48]

Tools currently on the market for the construction of lexical analyzers do not include procedures for the repair of lexical errors. For instance, LEX (one of the most popular systems operating under both DOS and UNIX)[49] only provides the user with the possibility of using a *default* clause, which acts just like any other word definition in the vocabulary. Any substring not recognized as a word belonging to the language is handled by procedures built into the clause in question.

1.2. Syntactic Errors. One of the objectives of a translator's syntactic analyzer is to check that input sentences are laid out according to whatever structure the particular language demands. Any deviation from the grammatical rules of a language L is deemed to be an error of syntax.[50−52]

If we consider a language L, defined by an alphabet of terminal symbols $\sum_T$ and where the language L contained all of the strings $\sum_T^*$, a syntax error could never be detected in an input sentence written in that language, since any combination of terminal symbols would be valid. Syntax errors could only be detected if the language L was a subset of $\sum_T^*$, and detection would be increasingly more effective as the size of the subset was reduced. However, the solution to this problem is not quite so simple. A language based on a very small subset of $\sum_T^*$ would only serve to increase the size of the sentences needed to represent a specific quantity of information, since the number of phrases would rise while the ability of the sentence to represent the knowledge would decrease.

Most grammars that define natural and scientific languages (the latter including programming and scientific languages such as nomenclatures) are context sensitive, but syntactic analyzers are based on a definition of the language to be analyzed by means of context-free grammars whose definition includes a series of restrictions to represent context sensitivity.

Errors are thus detected in two ways. One method deals with errors corresponding to the definition of the context-free grammar. These are detected by a parsing algorithm relying on a series of tables (see Figure 1) which are generated based on the strict definition of the context-free grammar. Such errors are detected when the parsing algorithm finds an error string in the appropriate table. The parser then calls on routines to recover from and possibly repair the error. These routines have a special function depending on whether one is using an ascending (LR) or descending (LL) parser.

The other method deals with context-sensitive errors, which are harder to detect due to the lack of precision (the looser definition) of context-sensitive restrictions. These errors are closely bound to semantic errors and will be dealt with in a subsequent section.

There are many and varied general strategies that may be used by a syntactic analyzer to recover from an error, which, although widely used, have not achieved complete acceptance:[10−12]

Panic mode recovery: the simplest method, used by most parsers. It is based on a defined terminal symbol set known as *synchronization symbols*. When an error is detected the parser eliminates symbols from the input until it finds a synchronization symbol. The parser thus ignores a (sometimes very important) part of the input without checking for the presence of other errors that may appear between the first error detected and the synchronization symbol.

Sentence-level recovery: It is based on correction within the input sentence (e.g., by introducing a synchronization symbol) that allows the analysis process to continue.

Error production: based on increasing the definition of the context-free grammar by including possible errors. To achieve this requires an excellent awareness of possible errors, as it is the included error productions which are responsible for reporting, recovery, and repair. The problem then arises of an undue enlargement of the defined grammar and a correspondingly bulkier parser, all to control a relatively low number of errors.

Global correction: the minimum number of changes required to correct an input string. Algorithms have been proposed to find, for an incorrect input I and a grammar G, a syntax tree for a related string I', in such a way that the number of insertions, deletions, and changes made to morphemes in order to convert I to I' be the absolute minimum. The introduction of these algorithms is rather costly in the case of context-free grammar (LL, LR) based-parsers, and other lines of investigation into different grammar types are being explored for this kind of parser: research is being conducted into *context-free stochastic grammars, probabilistic grammars*, and, in particular, *n*- and *ng-grammars*.[28,53−56]

1.3. Semantic Errors. When we talk of the semantics of a language, we are dealing with the representation or meaning of the sentences well-built from that language. Basically, a translator is only capable of detecting syntax errors. The definition of a given language does not include its semantics, except in the case of a definition of language types (errors to be treated as syntactic). Semantics are defined as actions included in grammar-defined productions, so the translator will not initially be aware of semantic errors. These will only be discovered when the target language is being constructed.

While the developer may detect semantic errors by the inclusion of procedures within the semantic actions contained in the production rules, recovery from these errors is fairly complex, since when the error is detected the rule in which it has been detected will already have been reduced, due to the very nature of the semantic actions, and the error might not correspond to the declared symbols in the reduced rule since it may have arrived at such a point solely due to previously-reduced rules.

Take, for example, the sentence *trilithium sulfine*, when the user really meant to say *lithium sulfide*, and the rule was

salt-rule ::= *electropositive-group* root-symbol suffix
| . . . . . .
{associated semantic actions;};

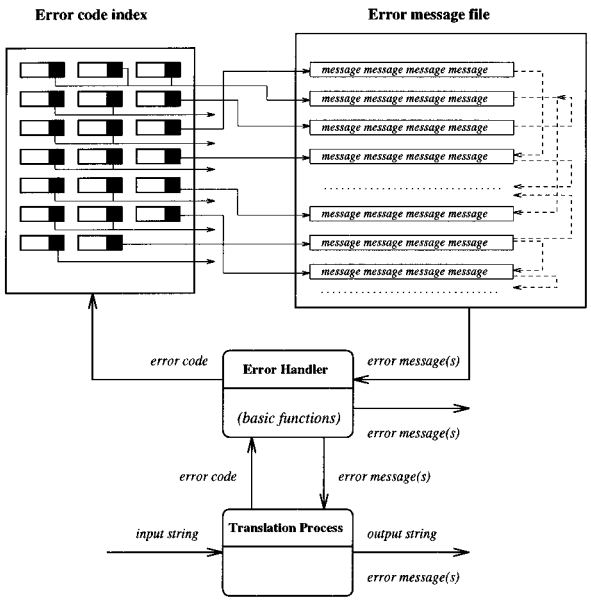

Figure 2. A general design for error reporting.

When the associated semantic actions are performed, a semantic error is detected, owing to the presence of the suffix *ine* instead of *ide* (or any other salt suffix). Moreover, the rule for the *Electropositive-group* metanotion has already been reduced, let us say a rule like

electropositive-group ::= multiplier element-name
| element-name oxidation-number
| root-symbol suffix
| . . . . . .

and no semantic error has been detected, due to the presence of an incorrect multiplier for charge balance. This error could only be detected by the semantic analyzer at the final stage of input string examination, because of the actions brought about by the previously-reduced rule (salt-rule).

1.4. Other Kinds of Errors. Errors produced in the generation of target-language sentences as well as system-generated errors will depend on not only the target language but also the quality of the translator and the environment in which it has been developed. The detection and handling of such errors is the duty of the developer. These errors should not directly depend on the translation process, although the complexity of the knowledge to be translated will obviously have a marked effect on the appearance of errors. Of course, the utilities and methods used in the development of the translator will also play a significant role in the successful functioning of the process.

1.5. Error Reporting. The user should always be informed of any errors detected by the translator and of the action taken; such error reports should use a language as close as possible to those of the real world.

A great deal of research has been carried out into the generation of user-friendly messages. In recent years, this research has been focussed on the development of report generators which will produce consistent, flexible messages in sentences close to the user's own language. This research is closely connected with work on explanatory systems in "intelligent" tutorials.[10,51,57,58]

The general notion of an error-reporting system might be viewed as follows (see Figure 2):

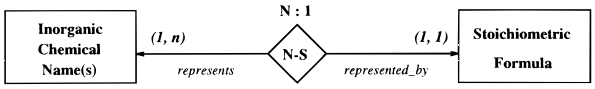

Figure 3. Relationship between name and formula in chemical substances.

- The developer should anticipate the errors that could occur during translation: lexical, syntactic, semantic, target-language generation, and execution errors.
- Errors should be classified according to the effect they might produce on the overall translation process and each of these should be assigned an appropriate identifying code.
- Suitable messages to be communicated to the user should be composed for each error or error type and stored in a table or error message file.
- There should be an index whose entry point is based on the error code and whose output accompanies one or more pointers to the appropriate message in the corresponding table.

The procedure of reporting to the user would then be as follows (see Figure 2): Given an error detected in the translation process to which a code had been assigned, reference would be made to the index and thereby to the message table; the user would be suitably informed and, according to the severity of the error, appropriate action would be taken.

Depending on its complexity, the error-reporting system may either output cryptic messages containing only an error code and a short message, or else provide the user with adequate information such as error type, string or substring in which the error occurred, any repair carried out or even whether the incorrect substring has been ignored, any correction that the user should make, and possible effects on target language sentences. All this should be reported in natural language.[7,57]

2. ERRORS IN CHEMICAL NAMES

In concept, the task of recognizing inorganic chemical names and generating an abstract or symbolic representation of the knowledge would seem trivial since, for example, there is a simple correspondence in the conversion of name to stoichiometric formula.[59]

The E-R diagram,[60] shown in Figure 3, illustrates how a name corresponds to one and only one formula, while, on the contrary, a chemical formula may be represented by more than one name (by more than one string in the same or different language).

However, the problem is far from trivial, as inorganic compound names may suffer from spelling or typing mistakes or the loss of part of the knowledge to be represented. Thus, a system that needs to recognize strings representing inorganic names for their subsequent translation into another representation (e.g., a data structure suitable for computer processing) should be equipped to deal with incorrect input strings containing any type of error in the chemical name that might prevent successful processing.

A suitable starting point for tackling the problem of error detection in the translation process would be the categorization of errors which might crop up in the text to be translated. Valuable work on the recognition of chemical names has been carried out by Eggert et al.,[59,61] who propose a classification of spelling and typing errors which perfectly

represents the situation a translator system may have to face, although their work is centered on the particular features of the translator which these authors are developing. Other studies directed at the recognition of chemical names in organic and inorganic chemistry make no special mention of error handling in the translation process.[1−3,5,62−68]

If errors are analyzed from the general point of view of a compiler called on to process source-language sentences and produce target-language sentences, it may be interesting to classify them thus as follows:

Type I: Errors not causing any loss in the knowledge represented by the original chemical name.

Type II: Errors producing some loss in the knowledge represented by the original chemical name.

Where the result of the translation process is concerned, only the second type of errors are of interest, as the first kind do not affect the end result and are only of interest when reporting back to the user.

Thus, the presence of extra spaces (*sulfuric_ _ _ _acid* instead of *sulfuric acid*), the absence of phonetic strings (*arsenous acid* instead of *arsenious acid*), the omission of oxidation numbers (*copper oxide* instead of *copper(II) oxide*), and so on will not affect the outcome, since the process does not break down; the detection of such errors is important, though, especially in the case of tutorial systems.

The translator system should be equipped to deal with these errors, so it will be necessary to increase the number of syntax rules defined in the parser in order to recognize all the different forms that may be used to represent an item of knowledge. It can be seen that since this type of error does not change the meaning of a chemical name, it will not affect the semantic analyzer which, in turn, need not be aware of the existence of such an error in a given input string.

Type II errors, however, lead to incorrect functioning of the translation process; we take correct translation to mean the accurate representation in the target language of the source language the user intended to input. For example, if the string *sodium sulfate* were introduced instead of *sodium thiosulfate*, the translator would not detect the error of omission (*thio*), since the input string is still a legal name.

Type II errors may be detected at any stage of translation: orthographic, lexical, syntactic, or semantic analysis, or they might even escape detection. They may be classified as follows:

Type II.1. *Detectable by the lexicographic analyzer*.

This group includes mechanical (typing) and spelling mistakes and is subdivided into the following:

Type II.1.1. *Typing errors*. Errors due to mechanical failure in the input phase, including the following:

1. Omission of characters (*ferric troxide* instead of *ferric trioxide*).
2. Insertion of characters (*nitric accid* instead of *nitric acid*).
3. Substitution of characters (which may be considered as an omission and an insertion) (*sodium chluride* instead of *sodium chloride*).
4. Transposition of characters (which may be considered as multiple omission and insertion) (*lithium hydrdie* instead of *lithium hydride*).

Type II.1.2. *Spelling mistakes*

Ignorance of the language (*uranil sulfate* instead of *uranyl sulfate*).

In practice, it is difficult to distinguish between type II.1.1 and type II.1.2 errors, since either way the input string appears to contain a typing error.

These errors all produce substrings which may contain more than one morpheme of the grammar and which will not then be recognized by the lexical analyzer. In the string *sodium sulafte*, for example, the erroneous substring is *sulafte* and contains two morphemes: the root-symbol morpheme *sulf* and the suffix morpheme *ate*. The transposition of the *f* and *a* characters means that neither of the morphemes is recognizable. The importance of this lies in the confusion arising when discussing "parts" of chemical names, since there are not usually any delimiters between morphemes, as described by Cooke-Fox.[62]

The absence of delimiters to separate morphemes (the words in the language's vocabulary) means that an error may affect several morphemes, so the error repair process should solve the following problems:

- It should find one or more morphemes to repair the error string and select the morpheme or combination of morphemes closest to the error string.
- It should perform whichever of the above possible repairs would introduce no further errors, bearing in mind that not every combination of morphemes is valid in the grammar as well as ensuring that the chosen combination is grammatically correct. The substitution of the error string by the repair string should produce a new input string which is at least grammatically correct in the current context.

Type II.2. *Detectable by the syntactical analyzer*.

Supposing that there are no type II.1 errors present, as they ought to have been detected at an earlier stage, the parser should detect illegal input sentence construction or, in other words, any excess, absence, or transposition of morphemes. These errors may be classified thus as follows:

1. Omission of morphemes (e.g., *sulfuric* instead of *sulfuric acid*).
2. Insertion of morphemes (e.g., *aluminum ditrioxide* instead of *aluminum trioxide*).
3. Transposition of morphemes (e.g., *dichloride carbonyl* instead of *carbonyl dichloride*).

As previously mentioned, the analyzer cannot always be guaranteed to recognize the above errors, for instance if the string *potassium sulfate* were entered in place of *potassium hydrogensulfate*, omitting the morpheme *hydrogen*.

Syntactic errors involve an incomplete knowledge of the language rather than a mechanical fault. Although they are easy to detect, they are sometimes difficult to repair, since this involves semantic considerations and more than one repair may seem suitable. Take the string *sulfur acid*, which contains a syntactic error of omission of the suffix morpheme:

- Either of the two suffixes *ic* and *ous* could perform a seemingly valid repair.
- For syntactic-semantic reasons, no other suffix (*ane*, *ite*, *ate*, *etc*.) could repair the error.

Another appropriate example would be the string *nitric*, which produces an omission error due the lack of the keyword token. Here, a larger number of tokens could be used to repair the string (*acid*, *oxide*, *anhydride*). Semantically, tokens such as *ion*, *cation*, and *anion* could not be employed, but there are no valid criteria to determine which of the list of possible tokens ought to be selected, or even

whether the possibility of multiple omission (prefix, radical, etc.) should be taken into account.

Type II.3. *Detectable by the semantic analyzer*.

As mentioned above, syntactic and semantic errors are closely related. The treatment of a type II.2 or type II.3 error strictly depends on the definition of the grammar used by the parser. Suppose the input string were *copper oxide* and the grammar relied on the following definition for the production of oxides:

oxide-rule ::= element-name oxidation-number *oxide*
{oxide recognition functions;};

In this case the syntactic analyzer would find an error due to the omission of the oxidation-number morpheme. It would be unable to reduce the previous production and would have to call up another routine to recover from the error and achieve repair, if possible. If, on the other hand, the definition of the grammar included the following productionthen the input string would not cause an error to be

oxide-rule ::= element-name oxidation-number *oxide*
{oxide recognition functions;}
| element-name *oxide*
{oxidation-number exception function;
oxide recognition functions;};

produced by the syntactic analyzer, as the grammar would provide a valid derivation. It is the semantic analyzer which detects the omission of the oxidation-number morpheme, and the developer must provide the corresponding report and repair functions. In the above case, "correct" repair would mean supposing that the oxidation number was the highest possible value for the element-name morpheme when acting as a metal.

Semantic errors are caused by ignorance of the laws of chemistry and not to factors which could create errors that would be detected by the other analyzers, and for that reason they have been classified as stated above.

One example of a semantic error might be the string *copper(V) oxide*, where the value of the oxidation-number morpheme violates the laws of chemistry, as *copper* cannot have a valency of +5. This kind of error is detected by the semantic functions defined alongside the production rules for the derivation of the string.

But let us examine another example, where the input string is *chlorate acid* and the production rule defined in the grammar is

acid-rule ::= root-symbol suffix *acid*
{acid recognition functions;};

In this case the rule is reduced if the string contains the root-symbol *(chlor)*, the suffix *(ate)* and the morpheme *acid*. Yet the string contains a semantic error, since only the suffixes *ic* and *ous* are valid. However, if the rule had been defined as then the error would have been detected by the parser.

acid-rule ::= root-symbol ic *acid*
{-ic acid recognition functions;}
| root-symbol ous *acid*
{-ous acid recognition functions;};

Expanding on this example, we could propose a generalization of the previous derivation rule thus and in this way the

general-rule ::= root-symbol suffix keyword

semantic analyzer would spot errors such as *sodate oxide*, *sodium anhydride*, *sodium anion*, etc.

The above example shows that if the rules of the parser are highly generic, the semantic analyzer becomes particularly complicated and vice versa, that is when they are highly specific then it is the definition of the parser's grammar which gets complicated. The interdependency between approaches to the treatment of syntactic and semantic errors is due to the fact that in the parser we are trying to represent a context-sensitive language (nomenclatures, in the present case) using a context-free language. In the parser, the extent of consideration afforded to context will affect to what degree errors are treated in each of the two categories.

3. DISCUSSION

This article has presented an overview of the various errors that may occur in the translation of knowledge from its external representation (that recognized by the user) to an internal format to be used by a computer program. While the literature abounds with research into the translation of chemical information, the detection and repair of errors has been largely neglected. Regardless of its nature, any translation process will include the stages described herein, so the development of error detection and repair procedures is crucial to the quality of such a system.

Lexicographic errors are those which may be treated in the most general form, since in any translator system the first step is to check that the morphemes of the input sentence are valid for the language. Accordingly, the next articles in this series will propose a general model for the repair of these errors.[69,70] This model is based on an approximate-matching algorithm that places particular emphasis on the structure and arrangement of the morphemes in chemical names.

The treatment of syntactic and semantic errors depends on the translator system, and varies with respect to the grammar defined and the nature of the parser. Since the utilities on the market for the construction of parsers work with context-free grammars (in both LR and LL parsers), error handling depends on the context-free grammar defined for each language and is normally context sensitive. Thus, it is the designer of the parser who decides, according to the extent of refinement of syntactic rules, whether error control is carried out by the syntactic or the semantic analyzer.

In a previous study,[58] the authors described a system for the translation of inorganic nomenclatures (IUPAC, Stock, and conventional) and of the trivial names of inorganic chemicals. This system incorporates an error handler in which the various types of errors are classified by severity and the stage at which they are detected; the translator is then informed of their presence. Although the system is capable of recovering from a great number of errors, it is still true that for this or indeed any translator system to be used in any area of science, there is a need for a powerful system to handle detection, recovery, and repair of errors. Such a system would be based on a model that could be applied to chemical names. One possible system is presented elsewhere.[69,70]

Although it is not difficult to repair and report illegal characters or type I errors (as classified in the present study),

the handling of type II errors would require specific procedures in order to supply valid repairs to the incorrect input substring via an approximate-matching process that would compare that substring with each and every morpheme allowed by the grammar. This is a nontrivial process since the repair process must not introduce any new errors. The repair should involve the production of a new, grammatically-correct string (i.e., a new name), and so the error-handler has to be aware of the language's valid grammatical structures. There exists, therefore, an interdependency between all the components of the language, lexis, syntax, and semantics, and even phonetics, as has been reported by other authors.[71,72]

REFERENCES AND NOTES

(1) Cooke-Fox, D. J.; Kirby, G. H.; Rayner, J. D. "Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 2. Development of a Formal Grammar." *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 106−112.

(2) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. "Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 3. Syntax Analysis and Semantic Processing." *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 112−118.

(3) Vander Stouw, G. G.; Elliot, P. M.; Isenberg, A. C. "Automated Conversion of Chemical Substance Names to Atom-Bond Connection Tables." *J. Chem. Doc.* **1974**, *14*(*4*), 185−193.

(4) Hippe, Z. *Artificial Intelligence in Chemistry*; Elsevier: Amsterdam, 1991.

(5) Warr, W. A. *Chemical Structures. The International Language of Chemistry*; Springer Verlag: Berlin, 1988.

(6) Bratchikov, I. L.; Diallo, M. M. "Elementary and Structural Language Errors and Methods for their Handling." *Automatic Control Comput. Sci.* **1991**, *25*(*4*), 81−84.

(7) Horning, J. J. *What the Compiler Should Tell the User Compiler Construction: An Advanced Course*; Springer-Verlag: 1976.

(8) Steegmans, E.; Lewi, J.; Van Horebeek, I. "Generation of Interactive Parsers with Error Handling." *IEEE Transactions Software Eng.* **1992**, *18*(*5*), 357−367.

(9) Yannakoudakis, E. J.; Fawthrop, D. "An Intelligent Spelling Error Corrector." *Information Process. Manage.* **1983**, *19*(*2*), 101−108.

(10) Fischer, C. N.; Leblanc, R. J. *Grafting a Compiler*; The Benjamin Cummings Publisher: 1988.

(11) Trembay, J. P.; Sonrenson, P. G. *The Theory and Practice of Compiler Writing*; McGraw-Hill: 1985.

(12) Aho, A. V.; Ullman, J. D. *The Theory of Parsing, Translation, and Compiling. Volumen I: Parsing*; Prentice-Hall: 1972.

(13) Segal, J.; Ahmad, K.; Rogers, M. "The Role of Systematic Errors in Developmental Studies of Programming Languages Learners." *J. Educ. Comput. Res.* **1992**, *8*(*2*), 129−153.

(14) Shaffer, L. H.; Hardwich, J. "Typing Performance as a Function of Text." *Qt. J. Exper. Psychol.* **1968**, *20*(*4*), 360−369.

(15) Bourne, C. P. "Frequency and Impact of Spelling Errors in Bibliographic Data Bases." *Inf. Process. Manage.* **1977**, *13*(*1*), 1−12.

(16) Mays, E.; Damerau, F. J.; Mercer, R. L. "Context Based Spelling Correction." *Inf. Process. Manage.* **1991**, *27*(*5*), 517−523.

(17) Morgan, H. L. "Spelling Correction in Systems Programs." *Commun. ACM* **1970**, *13*(*2*), 90−94.

(18) Pollock, J. J. "Spelling Error Detection and Correction by Computer: Some Notes and Bibliography." *J. Documentation* **1982**, *38*(*4*), 282−291.

(19) Pollock, J. J.; Zamora, A. "Collection and Characterization of Spelling Errors in Scientific and Scholarly Text." *J. Am. Soc. Inf. Sci.* **1983**, *34*(*1*), 51−58.

(20) Grefensfette, G. *Explorations in Automatic Thesaurus Discovery*; Kluwer Academic Publisher: 1994.

(21) Bentley, J. "A Spelling Checker." *J. CACM* **1985**, *28*(*5*), 456−462.

(22) Bickel, M. A. "Automatic Correction to Misspelled Names: A Fourth Generation Approach." *J. CACM* **1987**, *30*(*3*).

(23) Takahashi, H.; Itoh, N.; Amano, T.; Yamashita, A. "A Spelling Correction Method and its Application to an OCR System." *Pattern Recognition Lett.* **1990**, *23*, 363−377.

(24) O'Brien, P. "eL: using AI in CALL." *Intelligent Tutoring Media* **1992**, *3*(*1*), 3−21.

(25) Weinrich, K. B. "Unification-based Diagnosis of Language Learners Syntax Errors." *Literary Linguistic Comput.* **1991**, *6*(*3*), 149−154.

(26) Tappert, C. C.; Suen, C. Y.; Wakahara, T. "The State of the Art in On-Line Handwriting Recognition." *Pattern Anal. Machine Intelligence* **1990**, *12*(*8*), 787−808.

(27) Sinha, R. M.; Prasada, H. G.; Sabourin, M. "Hybrid Contextual Text Recognition with String Matching." *IEEE Trans. Pattern Anal. Machine Intelligence* **1993**, *15*(*9*), 915−925.

(28) Kukich, K. "Techniques for Automatically Correcting Words in Text." *ACM Computing Surveys* **1992**, *24*(*4*), 377−439.

(29) Bertossi, A. A.; Logi, F. "Parallel String Matching with Variable Length Don't Cares." *J. Parallel Distributed Computing* **1994**, *22*(*2*), 229−234.

(30) Idury, R. M.; Schaffer, A. A. "Dynamic Dictionary Matching with Failure Functions." *Theoretical Comput. Sci.* **1994**, *131*(*2*), 295−310.

(31) Colussi, L. "Fastest Pattern Matching in Strings." *J. Algorithms* **1994**, *16*(*2*), 163−189.

(32) Aho, A. V.; Corasick, M. J. "Efficient String Matching: An Aid to Bibliography Search." *J. CACM* **1975**, *18*(*6*), 333−340.

(33) Apostolico, A.; Giancarlo, R. "The Boyer-Moore-Galil String Searching Strategies Revisited." *SIAM J. Comput.* **1986**, *15*(*1*), 98−105.

(34) Boyer, R.; Moore, S. "A Fast String Matching Algorithm." *J. CACM* **1977**, *20*, 762−777.

(35) Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. *Introduction to Algorithms*; McGraw-Hill: 1990.

(36) Bunke, H.; Csirik, J. "An Algorithm for Matching Run-Length Coded Strings." *Computing* **1993**, *50*(*4*), 297−314.

(37) Tsay, Y. T.; Tsai, W. H. "Attributed String Matching by Split and Merge for Online Chinese Character Recognition." *IEEE Trans. Pattern Anal. Machine* **1993**, *15*(*2*), 180−185.

(38) Breslauer, D.; Galil, Z. "An Optimal O(log log N) Time Parallel String Matching Algorithm." *SIAM J. Comput.* **1990**, *19*(*6*), 1051−1058.

(39) Jokinen, P.; Tarhio, J.; Ukkonen, E. "Comparison of Approximate String Matching Algorithms." Helsinki University, Department of Computer Science, Technical Report: A-1991-7, 1991.

(40) Hall, P. A.; Dowling, G. R. "Approximate String Matching." *ACM Comput. Surveys* **1980**, *12*(*4*), 21.

(41) Marzal, A.; Vidal, E. "Computation of Normalized Edit Distance and Applications." *IEEE Trans. Pattern Anal. Machine* **1993**, *15*(*9*), 926−932.

(42) Wagner, R. A.; Fischer, M. J. "The String-to-String Correction Problem." *J. ACM* **1974**, *21*(*1*), 168−178.

(43) Ramalingam, G.; Reps, T. "An Incremental Algorithm for a Generalization of the Shortest-Path Problem." University of Wisconsin, Department of Computer Sciences, Technical Report, WI, 1992.

(44) Ukkonen, E. "Finding Approximate Patterns in Strings." *J. Algorithms* **1985**, *6*, 132−137.

(45) Tarhio, J.; Ukkonen, E. "Approximate Boyer-Moore String Matching." *SIAM J. Comput.* **1993**, *22*(*2*), 243−260.

(46) Baezayates, R. A.; Gonnet, G. H. "Fast String Matching with Mismatches." *Inf. Comput.* **1994**, *108*(*2*), 187−199.

(47) Crochemore, M.; Czumaj, A.; Gasieniec, L.; Jarominek, S.; Lecroq, T.; Plandowski, W.; Rytter, W. "Speeding-Up 2 String Matching Algorithms." *Algorithmica* **1994**, *12*, 247−267.

(48) Inui, M.; Shoaff, W.; Fausett, L.; Schneider, M. "The Recognition of Imperfect Strings Generated by Fuzzy Context-Sensitive Grammars." *Fuzzy Sets and Systems* **1994**, *62*(*1*), 21−29.

(49) Abraxas Software Inc., *PCYACC/PCLEX*., Abraxas Software Inc.: 7033 SW Macadam Ave., Portland, OR 97219, 1992.

(50) Holub, A. I. *Compiler Design in C*; Prentice-Hall: 1990.

(51) Lewis, P. M.; Rosenkrantz, D. J.; Stearns, R. E. *Compiler Design Theory*; Addison-Wesley: 1976.

(52) Martin, J. C. *Introduction to Languages and the Theory of Computation*; McGraw-Hill: 1991.

(53) Angell, R. C.; Freund, G. E.; Willet, P. "Automatic Spelling Correction Using a Trigram Similarity Measure." *Inf. Process. Manage.* **1983**, *19*(*4*), 255−261.

(54) Ullman, J. R. "A Binary n-gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words." *Comput. J.* **1977**, *20*(*2*), 141−147.

(55) Stolcke, A.; Seagal, J. "Precise n-gram Probabilities from Stochastic Context-free Grammars." University of California at Berkeley, International Computer Science Institute, Technical Report: TR-94-007, CA, 1994.

(56) Kuhn, R.; De Mori, R. "A Cache-Based Natural Language Model for Speech Recognition." *Pattern Anal. Machine Intelligence* **1990**, *12*(*6*), 570−583.

(57) Cawsey, A. *Explanation and Interaction*; M.I.T. Press: 1993.

(58) Luque Ruiz, I.; Cruz Soto, J. L.; Gómez-Nieto, M. A. "Computer Translation of Inorganic Chemical Nomenclature to a Dynamic Abstract Data Structure." *J. Chem. Inf. Comput. Sci.* **1994**, *34*(*3*), 526−533.

(59) Eggert, A. A.; Jacob, A. T.; Middlecamp, C. H. "Converting Chemical Names to Formulas. A Second Expert Problem." *J. Chem. Inf. Comput. Sci.* **1993**, *33*(*3*), 458−465.

(60) Chen, P. P. *Entity-Relationship Approach to Systems Analysis and Design*; North-Holland: 1980.

(61) Eggert, A. A.; Jacob, A. T.; Middlecamp, C. H. "Converting Chemical Formulas to Names. An Expert Strategy." *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 223−237.
(62) Cooke-Fox, D. I.; Kirby, G. H.; Rayner, J. D. "Computer Translation of IUPAC Systematic Organic Chemical Nomenclature. 1. Introduction and Background to a Grammar-Based Approach," *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 101−105.
(63) Weininger, D. "SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules." *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31−36.
(64) Mendez, B.; Moreno, J. A. "A Prolog Program for the Generation of the Molecular Formulas." *J. Chem. Educ.* **1990**, *67*(*3*), 234−235.
(65) Krishnan, S.; Krishnamurthy, N. S. "Compact Grammar for Algorithmic Wiswesser Notation Using Morgan Name." *Inf. Process. Manage.* **1976**, *12*, 19−34.
(66) De Jong, A. J. *In Chemical Structures: The International Language of Chemistry*; Springer Verlag: 1988.
(67) Gordon, J. E. "Chemical Inference 2. Formalization of the Languages of Organic Chemistry. Generic Systematic Nomenclature." *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 81−92.
(68) Conrow, K. "Computer Generation of Baeyer System Names of Saturated, Bridged, Bicyclic, Tricyclic and Tetracyclic Hydrocarbons." *J. Chem. Doc.* **1966**, *6*(*4*), 206−212.
(69) Luque Ruiz, I.; Cruz Soto, J. L.; Gómez-Nieto, M. A. "Error detection, recovery and repair in the translation of inorganic nomenclatures. 3. An Error Handler." *J. Chem. Inf. Comput. Sci.* Submitted for publication.
(70) Luque Ruiz, I.; Cruz Soto, J. L.; Gómez-Nieto, M. A. "Error detection, recovery and repair in the translation of inorganic nomenclatures. 2. A Proposed Strategy." *J. Chem. Inf. Comput. Sci.* **1996**, *36*, 16−24.
(71) Ullman, S. *Meaning and Style*; Oxford Blackwell: 1979.
(72) Muñoz Muñoz, J. M. "Criterios para el Diseño de un Programa de Gestión de una Base de Datos Léxica." Universidad de Córdoba, Departamento de Filologias Francesca e Inglesa y sus Didácticas, Tésis Doctoral, Córdoba, España, 1994.

CI9502212