IA

stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 2/73 FEBRUARY

IA

L.G.L.Th. MEERTENS and J.C. VAN VLIET

REPAIRING THE PARENTHESIS SKELETON OF ALGOL 68 PROGRAMS

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Abstract

The present report deals with one of the concrete aspects of the syntax of ALGOL 68, to wit, the parenthesis structure. Upon encountering an error in a piece of source text, good resynchronization of the parser is only then possible if it is known beforehand which opening parentheses are, and which are not, accompanied by a matching closing parenthesis (and vice versa). This holds especially for some parentheses, such as the quote-symbol, that have one same representation for opening and closing parenthesis.

In this report, two algorithms for repairing incorrect parenthesis skeletons are given. The first algorithm deals with "state switchers", i.e., parentheses enclosing sequences of symbols that lack syntactical structure (the inside of comments, pragmats and string-denotations). The second one deals with "braces", i.e., parentheses that have to occur nested (such as open - and close-symbols).

0. Preface

The degree in which compilers for languages like ALGOL 68 are able to recover from errors in the source text and to give meaningful error messages, i.e., error messages which are interpretable for the human programmer, varies considerably in practice. In the current effort undertaken at the Mathematical Centre to construct a machine-independent ALGOL 68 compiler, one of the design objectives is to reach a relatively high level of error-recoverability. This objective sprouts forth from the following two considerations:

i. It is expected that a major application area for our compiler will be the processing of student programs, where the emphasis is on the correction of syntactical errors. We hope to minimize the number of runs required to make the program syntactically correct.

ii. The generality of ALGOL 68, both on the abstract and the concrete aspects of syntax, tends to give rise to error messages which are difficult to interpret, unless this tendency is counteracted by a conscious effort.

The present report deals with one of the concrete aspects of the syntax of ALGOL 68*, to wit, the parenthesis structure.

If, somewhere in a piece of source text which starts with an opening parenthesis, an error occurs which causes the parser to be derailed, one

* At present, the definition of the Algorithmic Language ALGOL 68 is subject to a revision process. In this report, such changes as seem likely to be incorporated in the final revised Report have been taken into account. The main changes having a bearing on the subject of this report are the following:

i. The corresponding brackets of one construct must have the same style of representation. E.g., *if* *b* | *i* *else* *j)* is illegal. In fact, *(* and *if* are two different symbols.

ii. Corresponding comment-symbols must have the same representation. E.g., *¢ k = 0 #* is not a comment. On the other hand, comment-symbols with a different representation may be used as comment-items, so *¢ k = 0 # ¢* would be legal. In fact, *¢* and *#* are two different symbols.

may hope to use the closing parenthesis to bring it back in its track. Should, however, this closing parenthesis be missing (which might be the cause of derailment in the first place), then this strategy is not particularly helpful. It may appear that the solution would be to insert, as it were, the matching closing parenthesis in the source text when a different closing parenthesis is met, but then, if the source text contains an extra closing parenthesis, we are even worse off. The conclusion is that a good resynchronization of the parser is only then possible if it is known beforehand which opening parentheses are, and which are not, accompanied by a matching closing parenthesis (and vice versa). This holds especially for some parentheses, such as the *quote-symbol*, that have one same representation for opening and closing parenthesis. Therefore, it was decided to tackle this point in a radical way: the design of the first scan has been extended with the task to repair incorrect parenthesis skeletons.

1. Introduction

In the sequel, the term "parenthesis" will be used to denote a wider class of symbols than is usual: we shall use this term to stand for the following symbols, or rather, representations of symbols[*]:

"braces": $, (,), *begin*, *end*, [,], (/, /),
|, |:, *case*, *in*, *ouse*, *out*, *esac*,
if, *then*, *elsf*, *else*, *fi*, *for*, *from*,
by, *to*, *while*, *do*, and

"state switchers": ", ¢, #, *co*, *comment*, *pr*, *pragmat*.

The role played by the state switchers is so special as to warrant a separate treatment. Not only does one same symbol serve both as "opener" and as "closer" of certain constructions (as is also the case with the formatter-symbol), but, which is more important, the "item sequences" which are embraced by these symbols (the inside of comments, pragmats and string-denotations)lack syntactical structure and may contain braces in an arbitrary fashion that otherwise would have to occur "nested". Therefore, it is a hopeless task to treat the braces before it is known which parts of the program are, and which are not, item sequences, and, consequently, which braces have to be disregarded and which have to be taken into account. Moreover, such an item sequence may not contain another similar construction. (E.g., a comment may not contain another comment, although it may, possibly, contain the sequence of comment-items # *a* #, whereas format-denotations may contain other format-denotations.)

[*] Since it is, at this stage, neither possible nor necessary to make a distinction between, e.g., an open-symbol and a brief-case-start-symbol or a brief-condition-start-symbol, we shall use the paranotion open-symbol to stand for any of those. Moreover, where no confusion can arise, "symbol" will be used to indicate both symbols (in the sense of the ALGOL 68 report), and representations of symbols, indiscriminately.

Apart from this, the treatment of both types of parentheses runs largely in parallel. In general, errors in the parenthesis skeleton are "repaired" by marking a number of parentheses such that, by deleting these parentheses, a correct skeleton is obtained. E.g., the parenthesis skeleton

$$([ ] ([( )) [ )])$$

will be repaired by marking it thus:

$$([ ] (\overset{*}{[} ( )) [\overset{*}{)}]) \; .$$

This is not meant to imply that such parentheses will be disregarded completely in later stages, although they are bound to be handled differently from unmarked parentheses. Some additional means of reparation, based upon heuristic considerations, consist of adding or changing parentheses in the source text.

A principle to which we have strictly adhered is: no correct program will be "repaired". In fact, an even stronger version of this principle, to be formulated later on, applies to our repairing algorithm.

2. The treatment of state switchers

An ALGOL 68 program can be thought of as consisting of a sequence of (possibly empty) segments, separated by state switchers. To each of these segments a "state" may be assigned, which is either "neutral" or one of the state switchers. For a correct program, it is possible to assign these states in such a way that the first and the last segment are neutral and that at each state switcher we have a correct transition, i.e., the state switches to that state switcher if it was neutral and to neutral if the present state switcher is equal to the state, and otherwise the state is not affected. To give an example:

```
segments: ~~~~~~~ " ~~~ " ~~~ " ~~~~~~~ ¢ ~~~ co ~~~ " ~~~ ¢ ~~~~~~~
states  : neutral    "     "    neutral    ¢      ¢      ¢    neutral
```

(Note that the segments which have a state switcher as state are precisely those segments which are, or are contained in, an item sequence.) Obviously, if such an assignment of states is not possible, the program is incorrect. However, the transition of the state at some state switcher may be (locally) correct.

It is necessary to refine our definition of a correct transition slightly further. Although state switchers have one same representation for openers and closers, it is possible in some cases to derive from the context that a given state switcher, which then must be a quote-symbol, cannot be an opener or a closer. E.g., in the context of *d = "monday";* it can be shown that the first quote-symbol must be an opener and the second one a closer. Now, for a state switcher which has been shown to be a non-opener, the transition from the state neutral to the state *"* is not considered correct. A similar restriction applies to state switchers which have been shown to be non-closers.

The task of the algorithm for correcting the state-switcher skeleton can be formulated approximately as follows: assign states to each of the segments in such a way that the number of incorrect transitions is kept, in some sense, as low as possible. Before we are able to give a more accurate formulation, it is necessary to indicate what incorrect transitions and what interpretations, i.e., assignments of states to all of the segments, are admissible.

2.1. Admissible incorrect transitions

The elementary repairing actions consist of the marking of one state switcher, indicating that it should be disregarded in order to obtain a correct state-switcher skeleton. This implies that the state should not switch at such a state switcher. Therefore, for an incorrect transition to be admissible in such cases, it is necessary that the state does not change. Now, consider the following example with an obviously incorrect skeleton:

```
          ~~~~~ " ~~~~~ " ~~~~~ " ~~~~~ .
```

There exist three ways to repair this skeleton, each giving an interpretation with one incorrect transition.

```
          ~~~~~~~ " ~~~~~ " ~~~~~~~ "* ~~~~~~~  ;
state:    neutral    "    neutral   neutral

          ~~~~~~~ " ~~~~~ "* ~~~~~ " ~~~~~~~    ;
state:    neutral    "        "    neutral

          ~~~~~~~ "* ~~~~~~~ " ~~~~~ " ~~~~~~~  .
state:    neutral    neutral     "    neutral
```

Not each of these interpretations is equally desirable, for the following reason: The effect of assigning the state neutral to a segment is that it will be subject to syntactical analysis. If the corresponding segment in the "intended program" was neutral, this is obviously all right, but if it was (part of) an item sequence, this will probably give rise to some extraneous error messages. On the other hand, if a state other than neutral is assigned to a segment which in the intended program was neutral, the syntactical analysis of this segment will be omitted, with the possible result that some syntactical errors which otherwise would have been detected, will pass unnoticed, thus necessitating an extra run to detect these errors. In our philosophy this latter eventuality is considered far more undesirable than to require the programmer to ignore some error messages. Consequently, both the first and the last interpretation in our example are preferable to the middle one, where the states of both the left and the right segment in the incorrect transition are not neutral, and which, therefore, will not be admitted.

Another type of incorrect, but admissible, transition is found in those transitions which require a non-opener or a non-closer to be interpreted as an opener or closer, respectively.

The following criterion is obtained: an incorrect transition is admissible if and only if either

i. the state of both segments concerned is neutral, or

ii. the state switcher is a non-opener and the state switches from neutral to ″, or

iii. the state switcher is a non-closer and the state switches from ″ to neutral.

2.2. Admissible interpretations

An interpretation I is "locally admissible" if:

i. all transitions of I are either correct, or admissible incorrect transitions, and

ii. I assigns the state neutral to the first and the last segment.

An interpretation I is admissible if:

i. it is locally admissible, and

ii. there does not exist another locally admissible interpretation J, all of whose transitions are either correct, or are the same as the corresponding transition of I.

E.g., although the following interpretation is locally admissible,

```
           ~~~~~~~ ¢* ~~~~~~~ ¢* ~~~~~~~ ¢* ~~~~~~~ ,

state:     neutral    neutral    neutral    neutral
```

it is not admissible, because of the existence of the following two (admissible) interpretations:

```
            ~~~~~~~ ¢ ~~~~~ ¢ ~~~~~~~ ¢* ~~~~~~~ , and

state:      neutral    ¢     neutral    neutral


            ~~~~~~~ ¢* ~~~~~~~ ¢ ~~~~~ ¢ ~~~~~~~ .

state:      neutral    neutral     ¢    neutral
```

Clearly, there always exists at least one admissible interpretation. This can be shown, using the following argument:

i. There exists at least one locally admissible interpretation, to wit, the one which assigns the state neutral to each segment.

ii. Some locally admissible interpretation not being admissible implies the existence of another locally admissible interpretation with a smaller number of incorrect transitions. Since this number can, obviously, not be less than zero, by *reductio ad absurdum* our claim follows.

If the state-switcher skeleton was correct to begin with, there is only one admissible interpretation, that in which each transition is correct.

2.3. Comparing admissible interpretations

In the general case, there will be more than one admissible interpretation for a given sequence of segments. The problem is, therefore, to give a criterion according to which one of these interpretations can be chosen as, hopefully, the best. A simple criterion would be to count the number of incorrect transitions. We have chosen, however, for a more sophisticated criterion, mainly because all too often the same number of errors will be found for different interpretations. For example, for the state switcher skeleton

```
       ~~~~~~~¢~~~~~~~¢~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~
```

there are five admissible interpretations, each of which contains two errors:

```
                                                      *       *
       ~~~~~~~¢~~~~~~~¢~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~ ,

state: neutral    ¢    neutral    ¢       ¢    neutral neutral neutral

                              *                               *
       ~~~~~~~¢~~~~~~~¢~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~ ,

state: neutral    ¢    neutral neutral    #       #    neutral neutral

              *                       *
       ~~~~~~~¢~~~~~~~¢~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~ ,

state: neutral neutral    ¢    neutral neutral    ¢       ¢    neutral

                              *       *
       ~~~~~~~¢~~~~~~~¢~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~ ,

state: neutral    ¢    neutral neutral neutral    ¢       ¢    neutral
```

and

```
              *                                               *
       ~~~~~~~¢~~~~~~~¢~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~#~~~~~~~¢~~~~~~~

state: neutral neutral    ¢    neutral    #       #    neutral neutral
```

Rather than having all incorrect transitions weigh equally heavily, different "error values" have been assigned to the various types of incorrect transitions, based upon estimates of the likelihood of these transitions. Moreover, to some extent the symbols of which the segments consist are taken into account and compared with the state assigned to the segment, the idea being that, e.g., *begin* is more likely to occur in a neutral segment, than in other segments.[*]

In order to obtain estimates of the likelihood, the following, admittedly oversimplified, model has been used: There exists a universe of intended programs, having certain statistical properties (such as the mean length of a neutral segment). A source text is obtained in two steps. First, an intended program is drawn from the universe. Second, this program is subjected to a perturbation process, consisting of omission of glyphs or of replacement by other glyphs, at random.[**] Given the a priori distribution of intended programs and the statistical properties of the perturbation process, Bayesian analysis makes it possible, for a given source text, to derive the a posteriori probability that the source text was obtained from a certain intended program.

To illustrate the line of thought, a simple example, taken from natural language, may be useful. Take the following sentence:

s1: *A lold mehal was awarmed.*

Obviously, this is meaningless. We may assume something meaningful was intended, but that printing errors have garbled the message. What now was the intended message? There are several possible candidates. Suppose that, after having eliminated several of them, such as:

A bold metal was alarmed. ,

[*] The introduction of this type of argument might open the door for the clearly undesirable situation where a correct but very unlikely piece of program is "repaired" into a more likely one. This is precluded, however, by our definition of "admissible interpretation". The obvious advantage is the possibility to resynchronize when, for instance, one comment-symbol has disappeared in a piece of program richly supplied with comments.

[**] "Glyphs" are the atomic elements from which representations are built up, for example, the characters of a given character set.
The case of insertion of glyphs is not considered here, since the resulting symptoms can be ascribed to the (much more likely) replacement. E.g., the intended program corresponding to the source text *(1¢)* could have been *(1)*, but could equally well have been *(10)*, *(11)*, etc.
Likewise, the case of interchanging glyphs is not taken into consideration. Such errors rarely influence the parenthesis skeleton; also, they give rise to symptoms, if any, that can be explained in terms of replacement.

on the ground of their being ungrammatical or meaningless, we are left with two candidates:

s2: *A cold meal was warmed.* , and

s3: *A gold medal was awarded.*

Now suppose that we know the a priori probability, presumably based on the context, that the intended message was s2 or, alternatively, s3. We shall denote these by P(s2) and P(s3), respectively. If we also know the probabilities P(s1|s2) and P(s1|s3) that, due to printing errors, s2 and s3, respectively, will be turned into s1, we can compute the a posteriori probabilities that s2 or s3 were intended, given the fact that we have received s1, by the formulae:

$$P(s2|s1) = \frac{P(s1|s2)\ P(s2)}{P(s1|s2)\ P(s2) + P(s1|s3)\ P(s3)} \text{, and}$$

$$P(s3|s1) = \frac{P(s1|s3)\ P(s3)}{P(s1|s2)\ P(s2) + P(s1|s3)\ P(s3)} .$$

P(s1|s2) and P(s1|s3) can be determined by taking the product of the conditional probabilities for the individual printing errors (which are considered independent). So

$$P(s1|s2) = P(c \rightarrow l)\ P(+\ h)\ P(+\ a)\text{, and}$$

$$P(s1|s3) = P(g \rightarrow l)\ P(d \rightarrow h)\ P(d \rightarrow m).$$

The latter probabilities may be estimated by collecting statistical data on printing errors. Observe that, typically, the formulae for P(s2|s1) and P(s3|s1) have their denominators in common, so that, in order to compare them, it suffices to compare their numerators.

In our situation, we have no a priori ground to consider one intended program more likely than another one. Therefore, we will assume a uniform distribution of the intended programs*. For the above example, this would imply P(s2) = P(s3). Again, as our goal is comparison only, we obtain a simplification: it now suffices to compare the product of the conditional probabilities for the individual phenomena observed. Rather than multiplying probabilities and comparing products, we shall add error values, i.e., (scaled) logarithms of the probabilities and then compare the sum.

2.4. Determination of error values

As the model on which our algorithm is based is a gross simplification, no other significance should be attached to the error values obtained than that of a heuristic guide, obtained by making educated quesses.

In the subsequent sections, the following will be used to stand for (estimates of) probabilities and other parameters:

N = size of the glyph set,

L = size of the letter set,

ϕ_- = probability that a given glyph is omitted in the perturbation process,

ϕ_0 = probability that a given glyph is replaced by some other glyph,

S_Σ = the range over which the length of segments with state Σ may vary in practice, where Σ stands for some state switcher,

ϕ_b = probability that one or more bold symbols occur in a segment other than neutral.

* This is not wholly true, as we want to consider a program containing, e.g., the string "*begin*" less likely than the program with that string replaced by "*begin*". By means of a trick, however, we can save the uniform distribution. This can be described in terms of our model by restricting the universe of intended programs to programs that do not contain such strings, and than having these strings appear in the perturbation process.

2.4.1. Error values for comment-symbols

Where in the sequel C is used, this should be understood as standing for any of the state switchers ¢, #, *co*, *comment*, *pr* or *pragmat*.

Consider two segments with states Σ_1 and Σ_2, separated by C:

$$\underset{\Sigma_1}{\sim\sim\sim\sim\sim\sim}\; C \;\underset{\Sigma_2}{\sim\sim\sim\sim\sim\sim}\;.$$

In order that an interpretation which assigns Σ_1 and Σ_2 to these segments be admissible, it is necessary that at C we have either a correct or an admissible incorrect transition. So we must have one of the following (classes of) transitions:

i. Σ_1 = neutral, $\Sigma_2 = C$.

ii. $\Sigma_1 = C$, Σ_2 = neutral.

iii. $\Sigma_1 = \Sigma_2 = \Sigma$, where $\Sigma \neq C$ and $\Sigma \neq$ neutral.

iv. $\Sigma_1 = \Sigma_2$ = neutral.

It now remains to be seen how these cases can arise by subjecting an intended program to the perturbation process. In transitions i, ii and iii the intended program was not affected, as the transition is correct there. Although the probability of this phenomenon is slightly less than 1, as there is a small, but finite, probability that some perturbation will occur, we shall equate this probability to 1. In case iv, an incorrect transition is involved, and there is a multitude of possibly intended programs that could result in this phenomenon. In order to restrict the analysis to local effects, we shall assume that this incorrect transition is the only one in the interpretation of the source text. The given source text could have resulted from the replacement of some glyph in the intended program by C, but in the light of the much likelier possibility that the corresponding C was omitted or mistyped, we shall disregard this possibility. So the class of possibly intended programs comprises those programs that can be transformed into the source text by deleting one occurrence of C, or replacing it by some other glyph (more specifically, by the corresponding glyph in the source text). For each of these, we have: $P = \phi_- + \frac{\phi_0}{N}$. The number of possibly intended programs can be estimated roughly to be $2S_C$, one for each of the about S_C places to the left and to the right of C in the source text where the deleted or replaced C could have stood.

To summarize:

$$P(i) = P(ii) = P(iii) = 1;$$

$$P(iv) = 2S_C(\phi_- + \frac{\phi_0}{N}).$$

The corresponding error values are obtained by taking the logarithm to some base of these probabilities.

2.4.2. Error values for quote-symbols

Previously, we have already mentioned the fact that it is sometimes possible to see whether a given quote-symbol is an opener or a closer. As we expect that in most cases string-denotations will consist of letters mainly, only the information provided by these symbols is taken into account. String-denotations may only be followed and/or preceded by letters when they occur within format-denotations. Note that at this stage it is not yet known whether a given string-denotation does or does not occur in a format-denotation: this can only be checked when the brace skeleton is known, which in its turn has to await the treatment of the state switcher skeleton. The set of letters which may occur in format-denotations consists of: *a*, *b*, *c*, *d*, *e*, *f*, *g*, *i*, *k*, *l*, *n*, *p*, *r*, *s*, *t*, *x*, *y* and *z*. (Note that *f* and *g* do not occur in format-denotations as defined in the original Report. However, it seems likely that they will be included in the revised version of the syntax for format-denotations.) In order to make the probability of recognizing quote-symbols as openers or closers as great as possible, we do not only consider one symbol, but a maximal sequence of letters, i.e., a sequence of letters preceded and followed by a non-letter. Let $\bar{\delta}$ stand for the set of letters mentioned above, and δ for the set of remaining letters, *h*, *j*, *m*, *o*, *q*, *u*, *v* and *w*. Then a $\bar{\delta}$-sequence is a, possibly empty, maximal sequence of letters, each letter belonging to $\bar{\delta}$. A δ-sequence is a maximal sequence of letters, at least one of which is an element of δ.(So a sequence of letters is either a $\overline{\delta}$-sequence or a δ-sequence.) To give an example: *asnailspace* is a $\overline{\delta}$-sequence, whereas *andromeda* is not. (If a space-symbol has some representation, e.g., $\underline{\cdot}$, it should of course also be included in δ.) In the sequel, Δ will stand for a δ-sequence, and $\overline{\Delta}$ for a $\overline{\delta}$-sequence. In this way, we can distinguish four cases, depending on the context of the quote-symbol: $\overline{\Delta}\ ''\ \Delta$, $\Delta\ ''\ \overline{\Delta}$, $\overline{\Delta}\ ''\ \overline{\Delta}$ and $\Delta\ ''\ \Delta$.

2.4.2.1. The case $\overline{\Delta}\ ''\ \Delta$

As in section 2.4.1., we must have one of the four (classes of) transitions i to iv (where C stands for $''$).

Transition i and iii involve no errors, so P(i) = P(iii) = 1.

Transition ii can arise by mutilation of an intended program where the context of the quote-symbol is $\overline{\Delta}\ ''\ \overline{\Delta}$. The $\overline{\Delta}$ at the right must be an empty sequence, except within format-denotations, but for the sake of simplicity we shall disregard the latter case. The change of $\overline{\Delta}$ to Δ can occur by:

- replacing the glyph after the quote-symbol by the letter from δ $(P = (N-L)\ \frac{\phi_0}{N})$;
- omitting the glyph after the quote-symbol $(P = (N-L)\ \phi_-)$.

Taken together, $P(ii) = (N-L)(\phi_- + \frac{\phi_0}{N})$.

Transition iv can be treated as in 2.4.1., but now there are only about $S_{''}$ places where the original quote-symbol could have stood since, clearly, the quote-symbol at hand is an opener.

To summarize: $P(i) = P(iii) = 1;$

$$P(ii) = (N-L)\ (\phi_- + \frac{\phi_0}{N});$$

$$P(iv) = S_{''}\ (\phi_- + \frac{\phi_0}{N}).$$

2.4.2.2. The case $\Delta\ ''\ \overline{\Delta}$

This case is the mirror image of the case $\overline{\Delta}\ ''\ \Delta$, considered in section 2.4.2.1.

We obtain: $P(ii) = P(iii) = 1;$

$$P(i) = (N-L)\ (\phi_- + \frac{\phi_0}{N});$$

$$P(iv) = S_{''}\ (\phi_- + \frac{\phi_0}{N}).$$

2.4.2.3. The case $\bar{\Delta}$ ″ $\bar{\Delta}$

This case is similar to the case treated in section 2.4.1., if we take C to stand for ″.

We have: $P(i) = P(ii) = P(iii) = 1;$

$$P(iv) = 2S_{''}\ (\phi_- + \frac{\phi_0}{N}).$$

2.4.2.4. The case Δ ″ Δ

This is only correct if transition iii is involved. For transition ii, the analysis of the case $\bar{\Delta}$ ″ Δ applies, as does that of the case Δ ″ $\bar{\Delta}$ for transition i. In the remaining case of transition iv, the intended program must have been one where instead of the quote-symbol at hand some other symbol stood ($P = N\phi_0 \frac{1}{N} = \phi_0$).

Summarizing: $P(i) = P(ii) = (N-L)\ (\phi_- + \frac{\phi_0}{N});$

$$P(iii) = 1;$$

$$P(iv) = \phi_0.$$

2.4.3. Numerical results

Guesstimates for the various parameters in this section were obtained independently from some colleagues and used to compute the various error values (where the negative logarithm to some base was taken). The results are listed on page 17.

2.4.4. The algorithm

The task of the algorithm can be stated thus: find among all admissible interpretations an optimal one, i.e., one with minimal total error value for the segments and transitions involved. Obviously, it is impractical to generate all admissible interpretations one by one, as their number will grow exponentially with the number of state switchers in the source text. By applying the principle of dynamic programming, however, it is possible to derive a practical algorithm.

Estimates:

N	64	64	64
L	27	27	27
ϕ_-	.0001	.00163	.0003
ϕ_+	.0002	.00163	.0020
S_C	20	50	100
$S_{''}$	20	17	10

Error values

C	transition				average
comment	i,ii,iii iv	0 11.3	0 6.8	0 7.0	0 8
$\bar{\Delta}$ '' Δ	i,iii ii iv	0 11.5 12.7	0 10.5 13.4	0 11.4 14.8	0 11 14
Δ '' $\bar{\Delta}$	i ii,iii iv	11.5 0 12.7	10.5 0 13.4	11.4 0 14.8	11 0 14
$\bar{\Delta}$ '' $\bar{\Delta}$	i,ii,iii iv	0 11.3	0 10.8	0 13.0	0 12
Δ '' Δ	i,ii iii iv	11.5 0 17.6	10.5 0 24.2	11.4 0 16.2	11 0 19
	ϕ_b				5

i: neutral → C
ii: C → neutral
iii: C' → C' ($C' \neq C$ or neutral)
iv: neutral → neutral

2.4.4.1. Application of the dynamic programming principle

Let the segments of a source text be numbered from 0 to n. Consider two admissible interpretations I1 and I2 that assign the same state Σ_i to the i-th segment. So we have

$$\text{I1:}\ \underset{\Sigma_0}{\underset{\sim\sim\sim}{(0)}}\ T_1\ \underset{\Sigma_1}{\underset{\sim\sim\sim}{(1)}}\ \dots\ \underset{\Sigma_{i-1}}{\underset{\sim\sim\sim\sim\sim}{(i-1)}}\ T_i\ \underset{\Sigma_i}{\underset{\sim\sim\sim}{(i)}}\ T_{i+1}\ \underset{\Sigma_{i+1}}{\underset{\sim\sim\sim\sim\sim}{(i+1)}}\ \dots\ \underset{\Sigma_{n-1}}{\underset{\sim\sim\sim\sim\sim}{(n-1)}}\ T_n\ \underset{\Sigma_n}{\underset{\sim\sim\sim}{(n)}}\ \text{and}$$

$$\text{I2:}\ \underset{\Sigma'_0}{\underset{\sim\sim\sim}{(0)}}\ T'_1\ \underset{\Sigma'_1}{\underset{\sim\sim\sim}{(1)}}\ \dots\ \underset{\Sigma'_{i-1}}{\underset{\sim\sim\sim\sim\sim}{(i-1)}}\ T'_i\ \underset{\Sigma_i}{\underset{\sim\sim\sim}{(i)}}\ T'_{i+1}\ \underset{\Sigma'_{i+1}}{\underset{\sim\sim\sim\sim\sim}{(i+1)}}\ \dots\ \underset{\Sigma'_{n-1}}{\underset{\sim\sim\sim\sim\sim}{(n-1)}}\ T'_n\ \underset{\Sigma'_n}{\underset{\sim\sim\sim}{(n)}}\ ,$$

where the T's stand for the transitions involved and the numbers of the segments are given between parentheses. Then, clearly, both of the following interpretations are also admissible:

$$\underset{\Sigma_0}{\underset{\sim\sim\sim}{(0)}}\ T_1\ \underset{\Sigma_1}{\underset{\sim\sim\sim}{(1)}}\ \dots\ \underset{\Sigma_{i-1}}{\underset{\sim\sim\sim\sim\sim}{(i-1)}}\ T_i\ \underset{\Sigma_i}{\underset{\sim\sim\sim}{(i)}}\ T'_{i+1}\ \underset{\Sigma'_{i+1}}{\underset{\sim\sim\sim\sim\sim}{(i+1)}}\ \dots\ \underset{\Sigma'_{n-1}}{\underset{\sim\sim\sim\sim\sim}{(n-1)}}\ T'_n\ \underset{\Sigma'_n}{\underset{\sim\sim\sim}{(n)}}\ \text{and}$$

$$\underset{\Sigma'_0}{\underset{\sim\sim\sim}{(0)}}\ T'_1\ \underset{\Sigma'_1}{\underset{\sim\sim\sim}{(1)}}\ \dots\ \underset{\Sigma'_{i-1}}{\underset{\sim\sim\sim\sim\sim}{(i-1)}}\ T'_i\ \underset{\Sigma_i}{\underset{\sim\sim\sim}{(i)}}\ T_{i+1}\ \underset{\Sigma_{i+1}}{\underset{\sim\sim\sim\sim\sim}{(i+1)}}\ \dots\ \underset{\Sigma_{n-1}}{\underset{\sim\sim\sim\sim\sim}{(n-1)}}\ T_n\ \underset{\Sigma_n}{\underset{\sim\sim\sim}{(n)}}\ .$$

In words: it is admissible to cross over at the i-th segment.

To give an example: from the two admissible interpretations

$$\underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(0)}}\ \text{¢}\ \underset{\text{¢}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(1)}}\ \text{¢}\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(2)}}\ \overset{*}{\text{¢}}\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(3)}}\ \overset{*}{\#}\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(4)}}\ \text{¢}\ \underset{\text{¢}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(5)}}\ \#\ \underset{\text{¢}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(6)}}\ \text{¢}\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(7)}}\ ,$$

and

$$\underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(0)}}\ \overset{*}{\text{¢}}\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(1)}}\ \text{¢}\ \underset{\text{¢}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(2)}}\ \text{¢}\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(3)}}\ \#\ \underset{\#}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(4)}}\ \text{¢}\ \underset{\#}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(5)}}\ \#\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(6)}}\ \overset{*}{\text{¢}}\ \underset{\text{neutral}}{\underset{\sim\sim\sim\sim\sim\sim\sim}{(7)}}\ ,$$

which both assign neutral to the third segment, we can derive yet two other admissible interpretations:

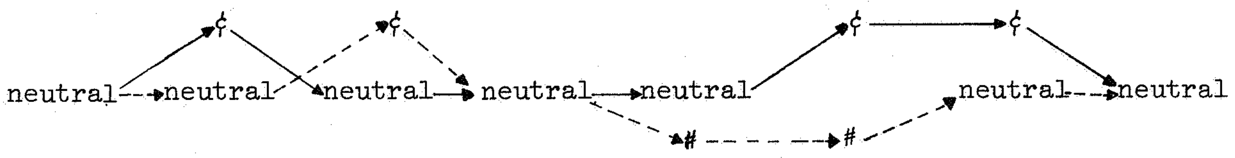
$$\underset{\text{neutral}}{\overset{(0)}{\sim\sim\sim\sim\sim\sim\sim}}\ \text{¢}\ \underset{\text{¢}}{\overset{(1)}{\sim\sim\sim\sim\sim\sim\sim}}\ \text{¢}\ \underset{\text{neutral}}{\overset{(2)}{\sim\sim\sim\sim\sim\sim\sim}}\ \overset{*}{\text{¢}}\ \underset{\text{neutral}}{\overset{(3)}{\sim\sim\sim\sim\sim\sim\sim}}\ \#\ \underset{\#}{\overset{(4)}{\sim\sim\sim\sim\sim\sim\sim}}\ \text{¢}\ \underset{\#}{\overset{(5)}{\sim\sim\sim\sim\sim\sim\sim}}\ \#\ \underset{\text{neutral}}{\overset{(6)}{\sim\sim\sim\sim\sim\sim\sim}}\ \overset{*}{\text{¢}}\ \underset{\text{neutral}}{\overset{(7)}{\sim\sim\sim\sim\sim\sim\sim}},$$

and

$$\underset{\text{neutral}}{\overset{(0)}{\sim\sim\sim\sim\sim\sim\sim}}\ \overset{*}{\text{¢}}\ \underset{\text{neutral}}{\overset{(1)}{\sim\sim\sim\sim\sim\sim\sim}}\ \text{¢}\ \underset{\text{¢}}{\overset{(2)}{\sim\sim\sim\sim\sim\sim\sim}}\ \text{¢}\ \underset{\text{neutral}}{\overset{(3)}{\sim\sim\sim\sim\sim\sim\sim}}\ \overset{*}{\#}\ \underset{\text{neutral}}{\overset{(4)}{\sim\sim\sim\sim\sim\sim\sim}}\ \text{¢}\ \underset{\text{¢}}{\overset{(5)}{\sim\sim\sim\sim\sim\sim\sim}}\ \#\ \underset{\text{¢}}{\overset{(6)}{\sim\sim\sim\sim\sim\sim\sim}}\ \text{¢}\ \underset{\text{neutral}}{\overset{(7)}{\sim\sim\sim\sim\sim\sim\sim}}.$$

This is illustrated perhaps more vividly in the following diagram, in which the interpretations are given by the paths from left to right. This way of looking at interpretations had proved a powerful heuristic guide.

$$\overset{(0)}{\sim\sim\sim\sim\sim\sim\sim}\ \text{¢}\ \overset{(1)}{\sim\sim\sim\sim\sim\sim\sim}\ \text{¢}\ \overset{(2)}{\sim\sim\sim\sim\sim\sim\sim}\ \text{¢}\ \overset{(3)}{\sim\sim\sim\sim\sim\sim\sim}\ \#\ \overset{(4)}{\sim\sim\sim\sim\sim\sim\sim}\ \text{¢}\ \overset{(5)}{\sim\sim\sim\sim\sim\sim\sim}\ \#\ \overset{(6)}{\sim\sim\sim\sim\sim\sim\sim}\ \text{¢}\ \overset{(7)}{\sim\sim\sim\sim\sim\sim\sim}$$



Now consider two partial interpretations

$$\text{P1: } \underset{\Sigma_0}{\overset{(0)}{\sim\sim\sim}}\ T_1\ \underset{\Sigma_1}{\overset{(1)}{\sim\sim\sim}}\ \ldots\ \underset{\Sigma_{i-1}}{\overset{(i-1)}{\sim\sim\sim\sim\sim}}\ T_i\ \underset{\Sigma_i}{\overset{(i)}{\sim\sim\sim}}\quad \text{and}$$

$$\text{P2: } \underset{\Sigma'_0}{\overset{(0)}{\sim\sim\sim}}\ T'_1\ \underset{\Sigma'_1}{\overset{(1)}{\sim\sim\sim}}\ \ldots\ \underset{\Sigma'_{i-1}}{\overset{(i-1)}{\sim\sim\sim\sim\sim}}\ T'_i\ \underset{\Sigma_i}{\overset{(i)}{\sim\sim\sim}}\quad ,$$

having their last states in common. Suppose that C1 is an optimal continuation of P1 resulting in an admissible interpretation, and, similarly, that C2 is an optimal continuation of P2. As stated above, C2 must be an admissible continuation of P1 also, and C1 of P2, so we have:

error value (P1C1) $\leq$ error value (P1C2) and

error value (P2C2) $\leq$ error value (P2C1).

Since the error value is obtained by adding together the error values of the individual transitions and segments, it is possible to apply the principle of dynamic programming. The above inequalities can be written thus:

error value (P1) + error value (C1) $\leq$ error value (P1) + error value (C2)
and
error value (P2) + error value (C2) $\leq$ error value (P2) + error value (C1).
From this, we derive: error value (C1) = error value (C2). Consequently,
error value (P1C1) < error value (P2C2) $\Longleftrightarrow$ error value (P1) < error value (P2).

In words: a partial interpretation P1 can only then beat another partial interpretation P2, i.e., can only then be the initial part of an interpretation which is better than the best possible completion of P2, if its error value is less than that of P2. As a consequence, only one out of P1 and P2 needs to be retained. This means that the set of partial interpretations of the segments 0 to i need contain at most one element for each of the (at most) eight possible states Σ_i for the i-th segment, to wit, an optimal partial interpretation. Initially, for i = 0, this set contains only one element, $\overset{(0)}{\underset{\sim}{\text{neutral}}}$. Scanning the state switcher skeleton from left to right, at each state switcher the set of partial interpretations is replaced by a new set. The transitions involved upon encountering a state switcher C can be depicted thus:

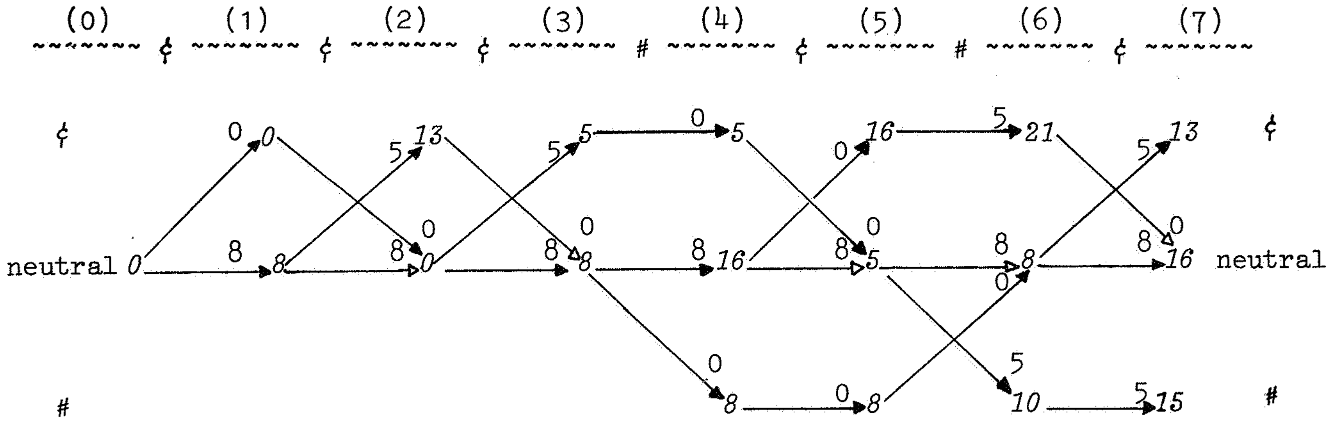
$$\begin{tikzcd}
\underset{\sim}{(i-1)} & C & \underset{\sim}{(i)} \\
\Sigma_{i-1} & & \Sigma_i \\
\text{neutral} \arrow[rr] \arrow[drr] & & \text{neutral} \\
C \arrow[urr] & & C \\
C' \arrow[rr] & & C'
\end{tikzcd}$$

$(C' \neq C)$

A decision has to be taken whetherthe new partial interpretation corresponding to the state neutral is the continuation of the old partial interpretation corresponding to neutral (P1) or ofthat corresponding to C (P2). This can be decided by comparing the sum of the error values of the old partial interpretation and the transition, i.e., by comparing

error value (P1) + error value (neutral $\overset{C}{\rightarrow}$ neutral) and
error value (P2) + error value (C $\overset{C}{\rightarrow}$ neutral).

After having processed the last segment of the source text, the (then complete) interpretation corresponding to the state neutral is an optimal interpretation. To give an example, which is slightly simplified in that only three states are involved, the state switcher skeleton which we already encountered in section 2.3. is displayed below (we assume that in segments 2, 3, 6 and 7 bold symbols occur), together with the correct or admissible incorrect transitions (drawn as arrows between states) and the corresponding error values. Where two transitions come together, one of them is drawn as an arrow with a white head, indicating that the partial interpretation corresponding to the other transition (drawn with a black head) is more profitable. The partial error values are displayed in italic font. By following the black-headed arrows back from the final state neutral, the optimal interpretation is obtained.



There is, however, one proviso. Although the interpretation obtained is certainly optimal among the locally admissible interpretations, there is no guarantee at all that, as a whole, it is admissible. Therefore, it is necessary to slightly amend the process sketched above.

2.4.4.2. Σ-admissibility

We define the notion "Σ-admissible" for partial interpretations in the following way:

Let P be a partial interpretation, assigning states to the segments 0 to i.
P is Σ-admissible (for short: P ∊ $\mathrm{Adm}_i(\Sigma)$) if

i. all transitions of P are either correct, or admissible incorrect transitions,

ii. P assigns the state neutral to segment 0 and Σ to segment i, and

iii. there does not exist another partial interpretation Q satisfying (i) and (ii), all of whose transitions are either correct, or are the same as the corresponding transition of P. (We say that such a Q "rules out" P.)

We can observe some facts, expressed in the form of a

Lemma:

(a) For a (total) interpretation I, the notions "admissible" and "neutral-admissible" coincide. Consequently, $\mathrm{Adm}_n(\text{neutral}) \neq \emptyset$.

(b1) $\mathrm{Adm}_0(\text{neutral}) = \{\underset{\text{neutral}}{\overset{(0)}{\sim\sim\sim\sim\sim\sim\sim}}\}$.

(b2) For $\Sigma \neq$ neutral, $\mathrm{Adm}_0(\Sigma) = \emptyset$.

(c) If $Q(\Sigma')\ T_i\ \underset{\Sigma}{\overset{(i)}{\sim\sim\sim}} \in \mathrm{Adm}_i(\Sigma)$, then $Q(\Sigma') \in \mathrm{Adm}_{i-1}(\Sigma')$.

(d) If P is ruled out from Σ-admissibility by some Q (not necessarily Σ-admissible itself), then there exists a Σ-admissible R ruling out P.

(e) If $P(\Sigma') \in \mathrm{Adm}_{i-1}(\Sigma')$ and T_i is a correct transition in the context of $\underset{\Sigma'}{\overset{(i-1)}{\sim\sim\sim\sim\sim}}\ T_i\ \underset{\Sigma}{\overset{(i)}{\sim\sim\sim}}$, then $P(\Sigma')\ T_i\ \underset{\Sigma}{\overset{(i)}{\sim\sim\sim}} \in \mathrm{Adm}_i(\Sigma)$.

Proof of (c):

From any R ruling out $Q(\Sigma')$ we can construct $RT_i\ \underset{\Sigma}{\overset{(i)}{\sim\sim\sim}}$, ruling out $Q(\Sigma')\ T_i\ \underset{\Sigma}{\overset{(i)}{\sim\sim\sim}}$.

Proof of (d):

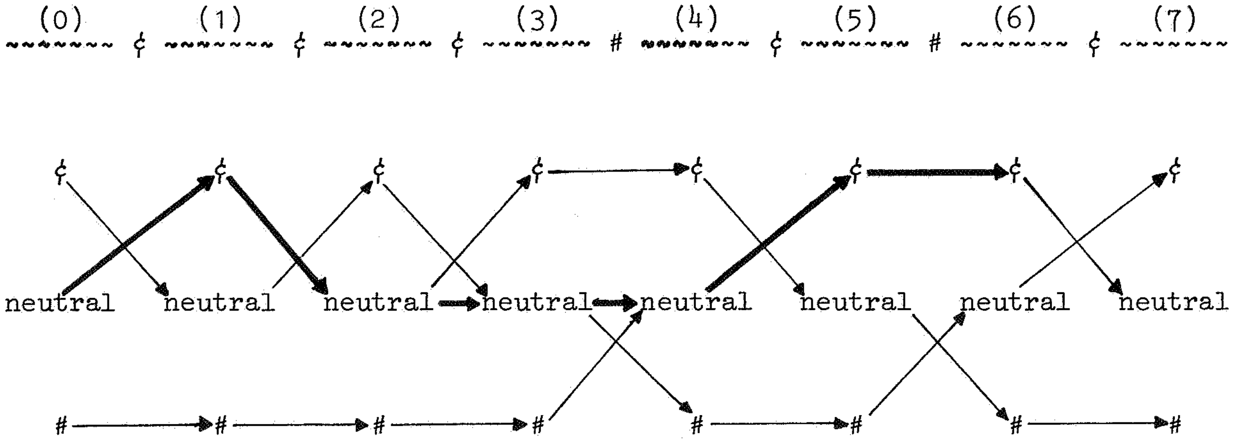
First, observe that "to rule out" is a transitive relationship. Moreover, if Q rules out P, then the number of incorrect transitions in Q is less than that in P. Therefore, a sequence of partial interpretations P, Q, Q', Q'', ..., each ruled out by the next, must have a finite length (in fact, cannot exceed the number of incorrect transitions in P by more than one). Consider such a sequence of maximal length P, Q, Q', Q'', ..., $Q^{(n)}$ (that is, no $Q^{(n+1)}$ exists ruling out $Q^{(n)}$). Then, $Q^{(n)}$ is such an R as we were looking for.

Proof of (e):

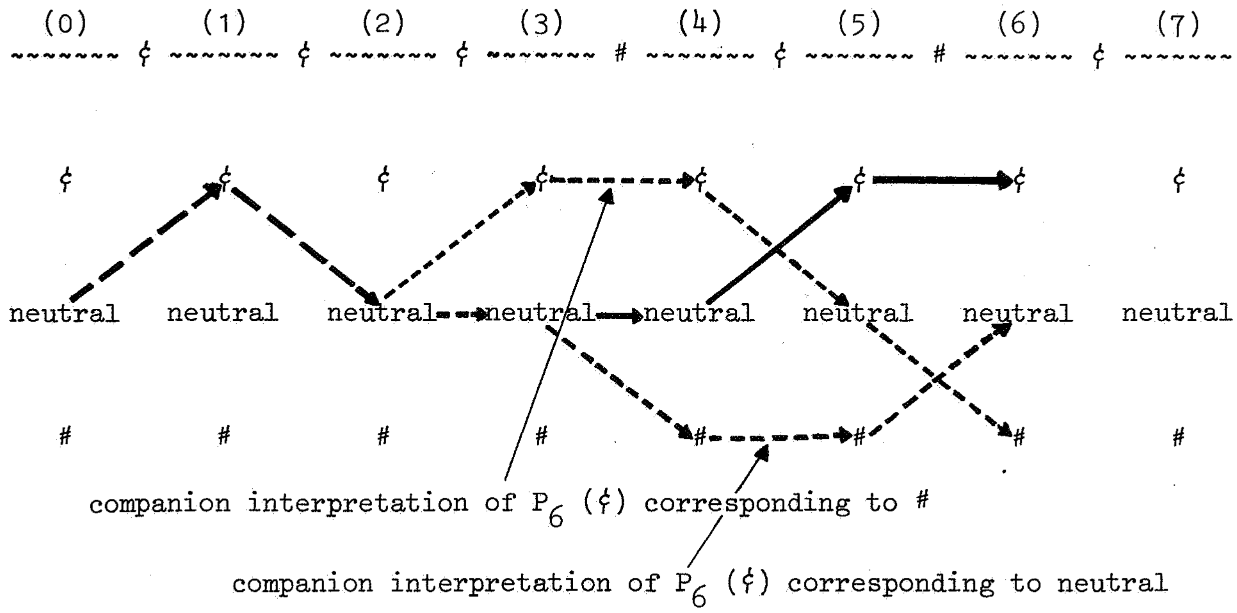
Suppose $P(\Sigma')\ T_i \overset{(i)}{\underset{\Sigma}{\sim\sim\sim}} \notin Adm_i(\Sigma)$, so that some $Q(\Sigma'')\ T_i \overset{(i)}{\underset{\Sigma}{\sim\sim\sim}}$ rules out $P(\Sigma')\ T_i \overset{(i)}{\underset{\Sigma}{\sim\sim\sim}}$. Now, assume $T_i \neq T_i'$. From the definition of "ruling out" we derive the correctness of T_i'. So we have $T_i \neq T_i'$, T_i is correct and T_i' is correct, simultaneously. However, for two different transitions to yield one and the same new state Σ, it is necessary that one of them be a transition neutral → neutral and therefore incorrect, which yields a contradiction. Evidently, the assumption was incorrect: $T_i = T_i'$. But then, $Q(\Sigma'')$ alone already rules out $P(\Sigma')$, contradicting $P(\Sigma') \in Adm_{i-1}(\Sigma')$.

2.4.4.3. The Companion Theorem

Given a partial interpretation $P_i(\Sigma)$ and some state Σ^*, we can informally define the "companion interpretation" of $P_i(\Sigma)$ corresponding to Σ^* thus: follow, in a diagram in which $P_i(\Sigma)$ is given by a path and in which the correct transitions are indicated, the correct transitions backwards starting from the state Σ^* at the i-th segment, until $P_i(\Sigma)$ is met, and then continue along $P_i(\Sigma)$. The partial interpretation thus obtained is the companion interpretation sought for. For example, in the following diagram



in which an interpretation P_6 (¢) is indicated by a path of bold arrows, we can obtain its companion interpretations corresponding to neutral and #, as displayed in the following diagram:



companion interpretation of P_6 (¢) corresponding to #

companion interpretation of P_6 (¢) corresponding to neutral

Such a companion interpretation need not always exist, since following the correct transitions backwards may either bring one to a point to which no correct transition leads, or the path thus obtained may never meet the given partial interpretation.

Furthermore, it is possible to distinguish two types of companion interpretations, in that one may or may not require that the last transition of the companion interpretation is a correct one. (This gives a distinction only if $\Sigma = \Sigma^*$ and the last transition of $P_i(\Sigma)$ is incorrect.) It is possible to give a more formal definition, by means of mutual recursion:

$$\mathrm{com}_{\Sigma^*}(P_i(\Sigma)) = \begin{cases} P_i(\Sigma) \text{ if } \Sigma^* = \Sigma \\ \\ \mathrm{Com}_{\Sigma^*}(P_i(\Sigma)) \text{ otherwise} \end{cases},$$

$$\mathrm{Com}_{\Sigma^*}(P_i(\Sigma)) = \begin{cases} \text{if } P_i(\Sigma) = P_{i-1}(\Sigma')\ T_i\ \underset{\Sigma}{\overset{(i)}{\sim\sim\sim}} \text{ and there exist } \Sigma^{**} \\ \text{and } T_i^* \text{ - which then are uniquely determined -} \\ \text{such that } \underset{\Sigma^{**}}{\overset{(i-1)}{\sim\sim\sim}}\ T_i^*\ \underset{\Sigma^*}{\overset{(i)}{\sim\sim\sim}} \text{ is a correct transition,} \\ \text{then } \mathrm{com}_{\Sigma^{**}}(P_{i-1}(\Sigma'))\ T_i^*\ \underset{\Sigma^*}{\overset{(i)}{\sim\sim\sim}}\text{, and, otherwise,} \\ \text{undefined.} \end{cases}$$

It is now possible to state the

Companion Theorem:

Let $P_{i-1}(\Sigma') \in \mathrm{Adm}_{i-1}(\Sigma')$, and $P_{i-1}(\Sigma')\ T_i\ \overset{(i)}{\widetilde{\Sigma}} \notin \mathrm{Adm}_i(\Sigma)$.
Then $\mathrm{Com}_\Sigma(P_{i-1}(\Sigma')\ T_i\ \overset{(i)}{\widetilde{\Sigma}})$ is defined and rules out $P_{i-1}(\Sigma')\ T_i\ \overset{(i)}{\widetilde{\Sigma}}$.

Proof:

Let $P_{i-1}(\Sigma')\ T_i\ \overset{(i)}{\widetilde{\Sigma}}$ be written as

$$\overset{(0)}{\widetilde{\Sigma_0}}\ T_1\ \overset{(1)}{\widetilde{\Sigma_1}}\ \cdots\ \overset{(i-1)}{\widetilde{\Sigma_{i-1}}}\ T_i\ \overset{(i)}{\widetilde{\Sigma_i}}\ ,$$

and let the partial interpretation ruling out $P_{i-1}(\Sigma')\ T_i\ \overset{(i)}{\widetilde{\Sigma}}$ be written as

$$\overset{(0)}{\widetilde{\Sigma^*_0}}\ T^*_1\ \overset{(1)}{\widetilde{\Sigma^*_1}}\ \cdots\ \overset{(i-1)}{\widetilde{\Sigma^*_{i-1}}}\ T^*_i\ \overset{(i)}{\widetilde{\Sigma^*_i}}\ .$$

$\Sigma^*_i = \Sigma_i$, but $\Sigma^*_{i-1} \neq \Sigma_{i-1}$, since, otherwise, $P_{i-1}(\Sigma')$ would be ruled out by

$$\overset{(0)}{\widetilde{\Sigma^*_0}}\ T^*_1\ \overset{(1)}{\widetilde{\Sigma^*_1}}\ \cdots\ \overset{(i-1)}{\widetilde{\Sigma^*_{i-1}}}\ .$$

Consequently, $T^*_i \neq T_i$. On the other hand, $\Sigma^*_0 = \Sigma_0$ (= neutral), so there must exist a maximal m, $0 \leq m < i$, such that $\Sigma^*_m = \Sigma_m$. From (e) we derive the incorrectness of T_i. On the other hand, according to the definition of ruling out, T^*_i must be correct. Also, for $m < n < i$, T^*_n must be correct, since, otherwise, we would have $T^*_n = T_n$ and, therefore, $\Sigma^*_n = \Sigma_n$, contradicting the maximality of m. Therefore, the following partial interpretation

$$\overset{(0)}{\widetilde{\Sigma_0}}\ T_1\ \overset{(1)}{\widetilde{\Sigma_1}}\ \cdots\ \overset{(m-1)}{\widetilde{\Sigma_{m-1}}}\ T_m\ \overset{(m)}{\widetilde{\Sigma_m = \Sigma^*_m}}\ T^*_{m+1}\ \overset{(m+1)}{\widetilde{\Sigma^*_{m+1}}}\ \cdots\ \overset{(i-1)}{\widetilde{\Sigma^*_{i-1}}}\ T^*_i\ \overset{(i)}{\widetilde{\Sigma^*_i}}$$

serves equally well to rule out $P_{i-1}(\Sigma')\ T_i\ \overset{(i)}{\widetilde{\Sigma_i}}$. But this is exactly the companion interpretation $\mathrm{Com}_\Sigma(P_{i-1}(\Sigma')\ T_i\ \overset{(i)}{\widetilde{\Sigma}})$, as is easily observed by inspection of the definition of Com.

2.4.4.4. Ensuring admissibility

We are now in a position to describe the amendment to the process: When it has to be decided for some $P_i(\Sigma)$ which of two candidates $P_{i-1}(\Sigma') \; T_i \; \overset{(i)}{\underset{\Sigma}{\sim\sim\sim}}$ and $P_{i-1}(\Sigma'') \; T_i^! \; \overset{(i)}{\underset{\Sigma}{\sim\sim\sim}}$ is chosen, then, before their error values are compared, a test is made to see if one candidate, say C, is ruled out by $Com_\Sigma(C)$, in which case only the other candidate is retained. We shall show that, for each i from 0 to n, the set of partial interpretations of the segments 0 to i thus obtained contains for each state Σ a partial interpretation $P_i(\Sigma) \in Adm_i(\Sigma)$, provided of course that $Adm_i(\Sigma) \neq \emptyset$.

It has to be shown:

(A) that, for i = 0, the set of partial interpretations fulfils the above requirement and

(B) that, for $i \geq 1$, it is possible to construct from such a set $\{P_{i-1}(\Sigma)\}$ a new set $\{P_i(\Sigma)\}$, also fulfilling the requirement, such that all of its elements are continuations of some element of $\{P_{i-1}(\Sigma)\}$, i.e., if $Adm_i(\Sigma) \neq \emptyset$, then $P_{i-1}(\Sigma') \; T_i \; \overset{(i)}{\underset{\Sigma}{\sim\sim\sim}} \in Adm_i(\Sigma)$ for some Σ' and some transition T_i.

From (A) and (B), together with the Companion Theorem, the claim then follows. For suppose that $P_i(\Sigma)$ has been constructed by the process as the one and only continuation of some $P_{i-1}(\Sigma')$ leading to the state Σ. According to (B), some Σ-admissible continuation exists; so, as there is only one continuation, it must be Σ-admissible. If, on the other hand, $P_i(\Sigma)$ has been chosen from two candidates, then these have been tested against their companion interpretation Com_Σ , so that, if they have been retained, they are, according to the Companion Theorem, Σ-admissible.

Proof of (A):

We have to show: if $Adm_0(\Sigma) \neq \emptyset$, then $P_0(\Sigma) \in Adm_0(\Sigma)$. This follows directly from the initial value of the set of partial interpretations, $\{\overset{(0)}{\underset{neutral}{\sim\sim\sim\sim\sim\sim\sim\sim}}\}$, combined with the facts (b1) and (b2) observed above.

Proof of (B):

Assume that, for some Σ, $Adm_i(\Sigma) \neq \emptyset$. $Adm_i(\Sigma)$ contains at least one element, which can be written as $Q(\Sigma') \; T_i \; \overset{(i)}{\underset{\Sigma}{\sim\sim\sim}}$. From (c), we see that

$Q(\Sigma') \in \mathrm{Adm}_{i-1}(\Sigma')$. $\mathrm{Adm}_{i-1}(\Sigma') \neq \emptyset$, so there exists, by hypothesis, a $P_{i-1}(\Sigma') \in \mathrm{Adm}_{i-1}(\Sigma')$. If $P_{i-1}(\Sigma')\; T_i\; \overset{(i)}{\widetilde{\Sigma}} \in \mathrm{Adm}_i(\Sigma)$, we are done. Suppose therefore $P_{i-1}(\Sigma')\; T_i\; \overset{(i)}{\widetilde{\Sigma}} \notin \mathrm{Adm}_i(\Sigma)$. By (d), this implies the existence of a $Q(\Sigma'')\; T'_i\; \overset{(i)}{\widetilde{\Sigma}} \in \mathrm{Adm}_i(\Sigma)$, ruling out $P_{i-1}(\Sigma')\; T_i\; \overset{(i)}{\widetilde{\Sigma}}$. Now, clearly, T_i and T'_i are two different transitions, since otherwise $Q(\Sigma'')$ alone would already rule out $P_{i-1}(\Sigma')$. Since $Q(\Sigma'')\; T'_i\; \overset{(i)}{\widetilde{\Sigma}}$ rules out $P_{i-1}(\Sigma')\; T_i\; \overset{(i)}{\widetilde{\Sigma}}$, T'_i must be a correct transition, according to the definition of "ruling out". Now, again applying (c) and the hypothesis, we infer the existence of a $P_{i-1}(\Sigma'') \in \mathrm{Adm}_{i-1}(\Sigma'')$. Using (e), we can construct $P_{i-1}(\Sigma'')\; T'_i\; \overset{(i)}{\widetilde{\Sigma}} \in \mathrm{Adm}_i(\Sigma)$.

By ensuring the admissibility of the interpretation obtained, we have, at the same time, lost the guarantee that the result will be optimal. This loss, however, does not worry us: sub-optimality can be shown to occur only as a consequence of having a non-vanishing error value for bold symbols within item sequences; setting these values equal to zero restores the guaranteed optimality. Since the error value in question has been chosen small compared to the other error values, a sub-optimal result, if at all, can only be so by a relatively small amount.

In order to be able to test the candidates against their companion interpretations, it is unnecessary to keep the $\mathrm{Com}_{\Sigma^*}(P_i(\Sigma))$, which would be cumbersome, or even information allowing their reconstruction. Instead, it suffices to remember, for each pair (Σ^*, Σ'), whether $\mathrm{com}_{\Sigma^*}(P_{i-1}(\Sigma'))$ is defined (or, for short, whether $\mathrm{def}_{i-1}(\Sigma^*,\Sigma')$). Then, $P_{i-1}(\Sigma')\; T_i\; \overset{(i)}{\widetilde{\Sigma}} \in \mathrm{Adm}_i(\Sigma)$ if and only if T_i is a correct transition or $\mathrm{def}_{i-1}(\Sigma^*, \Sigma')$ does not hold where Σ^* is the state, if any, such that T_i is correct in the context $\overset{(i-1)}{\widetilde{\Sigma^*}}\; T_i\; \overset{(i)}{\widetilde{\Sigma}}$.

The proof of this is left to the reader.

2.5. An example

The program given below is example 11.12 from the Report, in which three errors have been made:

in line 16 an *o* has been inserted before the close-symbol;
in line 25 a *"* has been inserted between the *o* and the *r*;
in line 38 $\underline{.}$ *"3* has been changed into *o "o*.

```
'BEGIN' 'MODE' 'RA' = 'REF' 'AUTH', 'RB' = 'REF' 'BOOK',
        'STRUCT' 'AUTH' = ('STRING' NAME, 'RA' NEXT, 'RB' BOOK),
                 'BOOK' = ('STRING' TITLE, 'RB' NEXT);
        'RA' AUTH,FIRST AUTH:= 'NIL',LAST AUTH; 'RB' BOOK;
        'STRING' NAME,TITLE; 'INT' I; 'FILE' INPUT,OUTPUT;
        OPEN(INPUT,,REMOTE IN); OPEN(OUTPUT,,REMOTE OUT);
        OUTF(OUTPUT,$P
               "TO ENTER A NEW AUTHER, TYPE ""AUTHOR"", A SPACE, AND HIS
                NAME."L
               "TO ENTER A NEW BOOK, TYPE ""BOOK"", A SPACE, THE NAME OF THE
                AUTHOR, A NEW LINE, AND THE TITLE."L
              "FOR A LISTING OF THE BOOKS BY AN AUTHOR, TYPE ""LIST"", A SPACE,
                AND HIS NAME."L
              "TO FIND THE AUTHOR OF A BOOK, TYPE ""FIND"", A NEW LINE, AND THE
                TITLE."L
               "TO END, TYPE ""END"""AL$,"."

INCORRECT TRANSITION; OLD STATE: " ; NEW STATE: NEUTRAL

                                            0));
        'PROC' UPDATE =
          : 'IF' ('RA': FIRST AUTH):=: 'NIL'
          'THEN' AUTH:= FIRST AUTH:= LAST AUTH:= 'AUTH':= (NAME,0,0)
          'ELSE' AUTH:= FIRST AUTH; 'WHILE' ('RA': AUTH):≠: 'NIL' 'DO'
                    (NAME = NAME 'OF' AUTH|KNOWN|AUTH:= NEXT 'OF' AUTH);
                    LAST AUTH:= NEXT 'OF' LAST AUTH:= AUTH:= 'AUTH':=
                    (NAME,0,0)) KNOWN: 'SKIP'
                'FI' # END DECLARATION PRELUDE SEQUENCE #;
CLIENT: INF (INPUT,$C("AUTHO"R"

INCORRECT TRANSITION; OLD STATE: NEUTRAL; NEW STATE: NEUTRAL

                               ,"BOOK","LIST","FIND","END",""),X30AL,
                       80AL$,I);
        'CASE' I 'IN' AUTHOR,PUBL,LIST,FIND,END,ERROR 'ESAC';
AUTHOR: IN (INPUT,NAME); UPDATE; CLIENT;
PUBL:  IN (INPUT, (NAME,TITLE)); UPDATE;
                 'IF' ('RB': BOOK 'OF' AUTH):=: 'NIL'
               'THEN' BOOK 'OF' AUTH:= 'BOOK':= (TITLE,0)
               'ELSE' BOOK:= BOOK 'OF' AUTH; 'WHILE' ('RB': NEXT 'OF' BOOK) :≠:
                  'NIL' 'DO'
                    (TITLE = TITLE 'OF' BOOK|CLIENT|BOOK:= NEXT 'OF' BOOK);
                 (TITLE ≠ TITLE 'OF' BOOK|NEXT 'OF' BOOK:= 'BOOK':= (TITLE,0))
                'FI'; CLIENT;
LIST:   IN (INPUT,NAME); UPDATE;
        OUTF(OUTPUT,$P"AUTHOR:O"

INCORRECT TRANSITION; OLD STATE: " ; NEW STATE: NEUTRAL

                                00ALL$,NAME);
          'IF' ('RB': BOOK 'OF' AUTH):=: 'NIL'
        'THEN' PUT(OUTPUT,"NO PUBLICATIONS")
        'ELSE' 'WHILE' ('RB':BOOK) :≠: 'NIL' 'DO'
                'BEGIN' 'IF' LINENUMBER(OUTPUT) = MAX LINE[REMOTE OUT]
                  'THEN' OUTF(OUTPUT,$41K"CONTINUED ON NEXT PAGE"P
                              "AUTHOR: "30A41K"CONTINUED"LL$,NAME)

                  'FI'; OUTF(OUTPUT,$80AL$,TITLE 'OF' BOOK);
                BOOK:= NEXT 'OF' BOOK
                'END'
        'FI'; CLIENT;
FIND:   IN (INPUT, ('LOC' 'STRING',TITLE)); AUTH:= FIRST AUTH;
        'WHILE' ('RA': AUTH) :≠: 'NIL' 'DO'
          'BEGIN' BOOK:= BOOK 'OF' AUTH;
                'WHILE'('RB': BOOK) :≠: 'NIL' 'DO'
                    'IF' TITLE = TITLE 'OF' BOOK
                'THEN' OUTF (OUTPUT,$L"AUTHOR: "30A$,NAME 'OF' AUTH); CLIENT
                'ELSE' BOOK:= NEXT 'OF' BOOK
                  'FI'; AUTH:= NEXT 'OF' AUTH
        'END';
        OUTF(OUTPUT,$L"UNKNOWN"L$,); CLIENT;
END:    PUT(OUTPUT,(NEWPAGE,"SIGNED OFF",CLOSE));
        CLOSE(INPUT).
ERROR:  PUT(OUTPUT,(NEW LINE,"MISTAKE, TRY AGAIN."));
        NEW LINE(INPUT); CLIENT
'END'
```

3. The treatment of braces

After, as a result of the treatment of the state switchers, states have been assigned to all of the segments, it is known which parts of the program are item sequences and, consequently, which braces have to be disregarded. The remaining braces constitute the brace skeleton. It is the task of the algorithm for repairing the brace skeleton to try to match as much as possible the braces and to mark the remaining ones. The criterion for comparing two alternatives will simply be: which one has the smaller number of marked braces.

First, it is necessary to define more precisely when two braces match. To begin with, the following pairs of braces match:

$ and $,　　(/ and /) ,
(and) ,　　*if* and *fi* ,
begin and *end* ,　　*case* and *esac* .
[and] .

In the case of, e.g., the brace |, the potential for matching is much higher: this brace is able to match simultaneously to the left with either of the braces (, | and |:, and to the right with |, |: and). In spite of this seeming complication, a satisfactory and yet simple treatment is given by systematically replacing | by)|(and |: by)|:(, and then to require the matching of (and), after which | and |: are no longer considered braces. To give an example, (||:|) would be replaced by ( )|( )|:( )|( ), thus giving a perfect matching of braces. Similarly, *then* is replaced by *fi then if*, and so on. Obviously incorrect skeletons, such as (|||)or *if else fi*, are not indicated as such by this algorithm; this task is delegated to a later scan.

Still another complication is presented by the braces *for from by to while do*. Here, the brace *by*, e.g., may match to the left with *for* or *from* or may not have a matching left brace at all. To the right it may match with *to*, *while* and *do*. This case is handled by the introduction of a virtual brace, matching *for*, which we shall denote here as *rof*. Using this, *to*, e.g., is either replaced by *to for* (as in the context *l*:*to*) or by *rof to for* (as in the context *for i to*). The first possibility has to be chosen when *to* is the first symbol of a loop, the second one when it is not the first symbol (and is, therefore, preceded by a unit). From a context free version of the Syntax it was possible to derive mechanically the set of symbols that may immediately precede a loop:

:=   :   =   (*begin*   ,   [*   (/*

@* | |:   *case*   *in*   *ouse*   *out*   *if*

then   *elsf*   *else*   ;   *from**   *by**   *to**   *while*   *do*,

and the set of symbols that may be the last symbol of a unit:

0...9   *true*   *false* $ void   )   *end*   ]   /)

esac   *fi*   *skip**   ~*   *nil**   o*   "   a...z

mode-indication.

As these sets are disjoint, it is always possible to decide if *to* should be replaced by *to for* or by *rof to for*, by inspecting one symbol immediately preceding *to*. Using these rules,

for i *to* 10 *while* *to* 3 *do* *while* b *do* s; c *do* t is replaced by

for i *rof* *to* *for* 10 *rof* *while* *for* *to* *for* 3 *rof* *do* *while* *for* b *rof* *do* s; c *rof* *do* t.

Formatter-symbols are somewhat special among the braces, in that they share with state switchers that opening brace and closing brace have one same representation. Similar to quote-symbols, it is sometimes possible to determine from the context whether a given formatter-symbol is an opening brace or a closing brace. In fact, there are only three contexts in which it is not possible to determine the nature of the formatter-symbol: ,$, , ,$+ and ,$-. This is due to the fact that a collection may be empty; if this were not the case, the decision could always be made.

* These symbols appear here due to the fact that the context free version encompasses a somewhat larger language.

Let Δ stand for the set of symbols that may be the beginning of a collection-list, but with the exception of the comma-, plus-, minus-symbol, since these may also immediately follow a format-denotation. So

$$\Delta = \{a, b, c, d, f, g, k, l, n, p, s, t, x, y, z, 0...9, ., (, "\}.$$

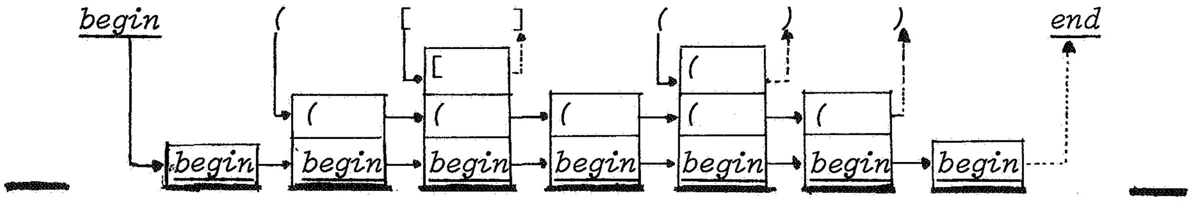
Let ∇ stand for the set of symbols that may be the end of a collection-list, but with the exception of the comma-symbol, which may also immediately precede a format-denotation. So

$$\nabla = \{a, b, d, g, k, l, p, t, x, y, z, .,), \underline{end}, \underline{esac}, \underline{fi}, "\}.$$

Let $\blacktriangle$ stand for the set of symbols that may not be the beginning, and $\blacktriangledown$ for the set of symbols that may not be the end of a collection-list. So $\Delta \cup \blacktriangle \cup \{,, +, -\} = \nabla \cup \blacktriangledown \cup \{,\}$ = the set of ALGOL 68 symbols. If we have $\$\ \Delta$ or $\blacktriangledown\ \$$ (e.g., $\$l$ or $=\$$), then $\$$ must be an opening brace. If we have $\nabla\ \$$ or $\$\ \blacktriangle$ (e.g., $)\$$ or $\$;$), then $\$$ must be a closing brace. If neither holds, then we must have one of the three undeterminable cases mentioned above; if, on the other hand, both possibilities hold (as in $)\$l$), then the source text contains an error.

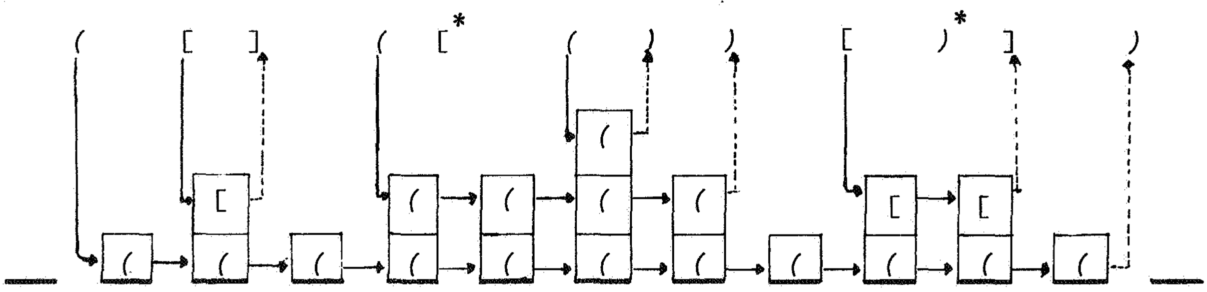
3.1. The algorithm

A correct brace skeleton, i.e., one in which all braces match properly, can be characterized algorithmically:
Start with an empty stack. Scan the braces from left to right. Upon meeting an opening brace, it is put on the stack. When a closing brace is met, it matches the brace on top of the stack (otherwise the brace skeleton was incorrect) and that brace is "matched away", i.e., removed from the stack top. After having processed all braces, the stack is again empty. To give an example (where the dotted line indicates a match):



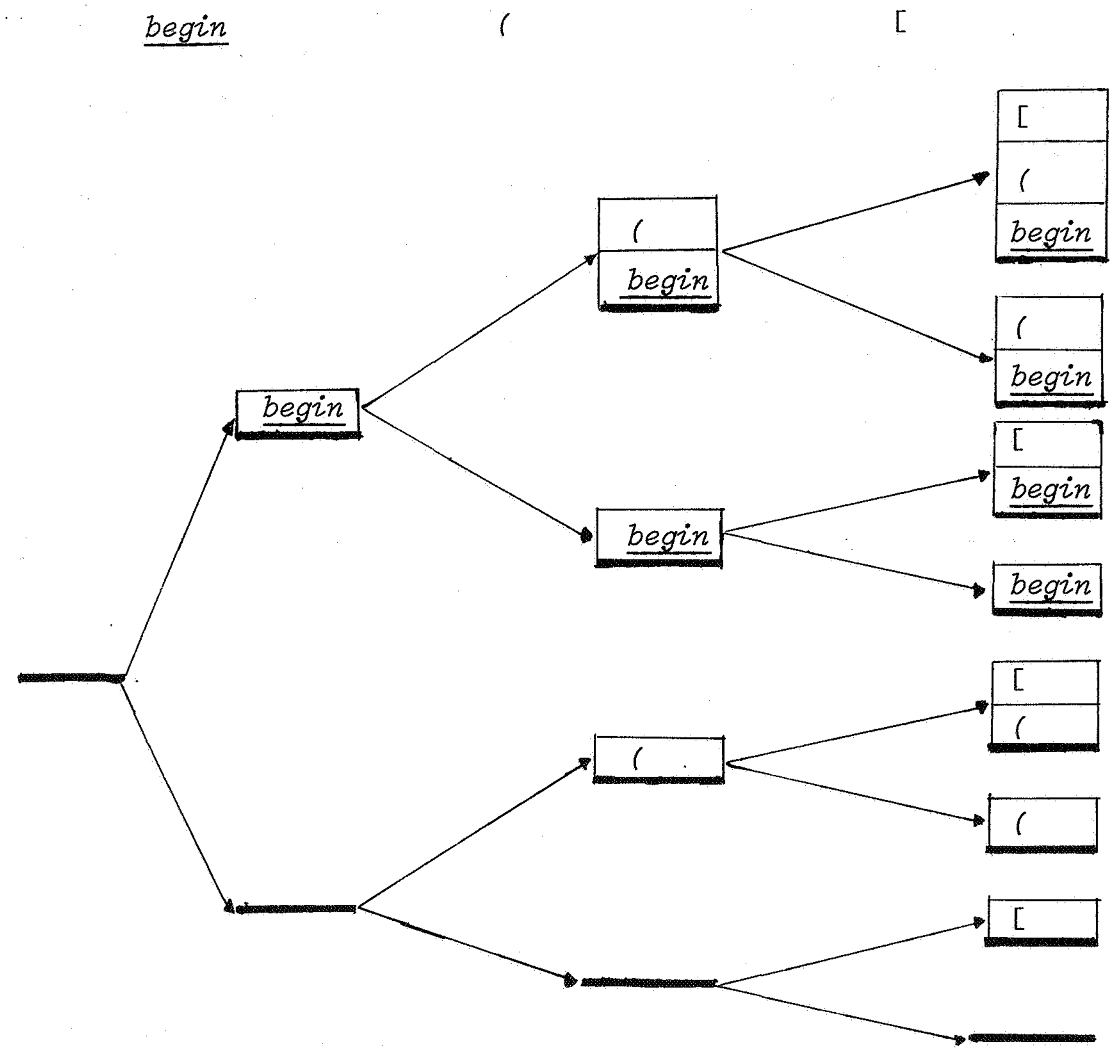
An alternative way of viewing this can bring the notion of "correct brace skeleton" in a framework quite similar to that employed for dealing with state switchers: The stacks are *states* that are assigned to the segments between the braces. For a correct program, it is possible to assign these states in such a way that the first and the last segment are neutral (i.e., have an empty stack) and that at each brace we have a correct transition, i.e., if the brace is an opening brace the new state consists of the old state with that brace put on top, and if it is a closing brace, the old state consists of the new state with the matching opening brace put on top.

The definition, derived from that of "repairing" as given in the introduction, is: an admissible incorrect transition is one in which the old and the new states are equal. In this way, the example given there would be treated as follows:

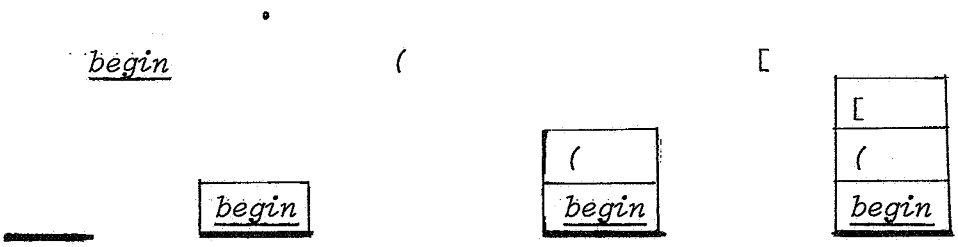


3.1.1. Growing a tree

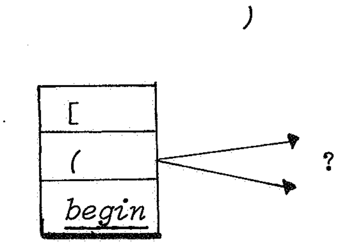
It is possible to derive from this definition a repairing algorithm in an obvious, straightforward way: the algorithm must grow, while scanning the brace skeleton from left to right, a tree whose nodes are states (i.e., stacks), such that each path from the root of the tree to a terminal node is a partial interpretation. But then, upon meeting an opening brace, the algorithm would have to grow from each terminal node two branches: one where the opening brace is put on top of the last state, and one where it is disregarded. The following example shows the doubling of the number of states that would thus occur at each opening brace:

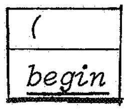


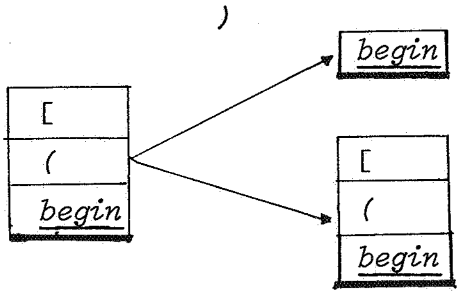
This is very impractical since, in striking contrast to the state switcher case, the number of potential states is unlimited (instead of being eight), so there is no hope of applying here the simple dynamic programming approach. Therefore, opening braces will always be put on top of the stack. A stack with three braces, say, represents eight states at once, one for each of the possible combinations obtainable by deleting none up to all of the cells. In the above example, the tree to be grown becomes linear:



but its meaning, in terms of states, is still the same as that of the forked tree above. The task of discovering that one or more cells of a stack should be deleted, is postponed and delegated to transitions involving closing braces. Consider, e.g., the following situation:

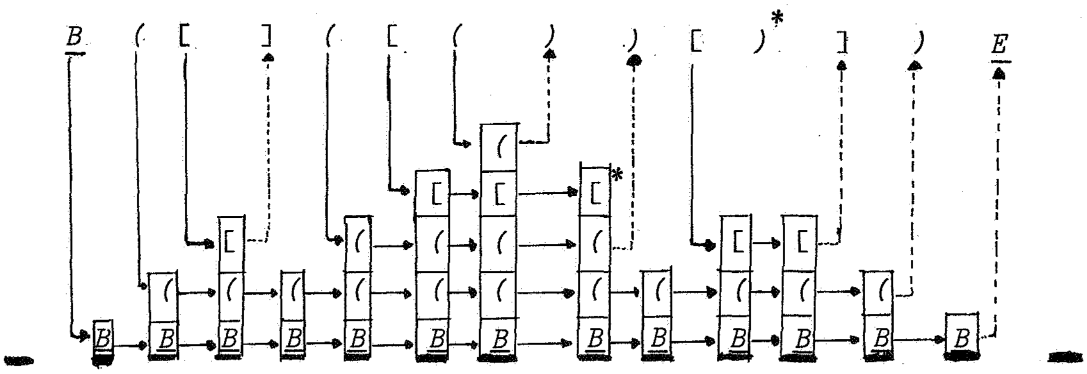


Remember that the stack at left represents, among other possibilities, the state  . Clearly, at this transition it is allowed to delete the top-cell in order to match away the underlying brace. Another possibility is to disregard the closing brace, so that we have two possible new stacks:



Using this approach, there would no longer exist an admissible interpretation for the brace skeleton ([], since no closing brace would remove the $\overset{*}{(}$ from the stack. Therefore, each brace skeleton is enclosed between two unique matching braces, denoted by *B* and *E*, before it is submitted to the algorithm.

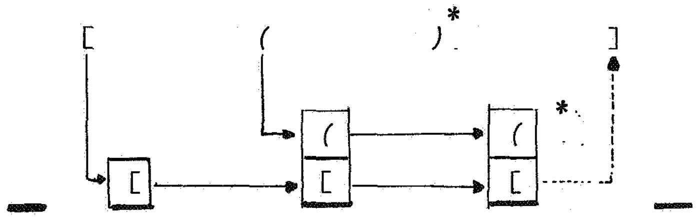
In the example from the introduction, we can obtain the following interpretation, where marked braces are indicated with an asterisk:



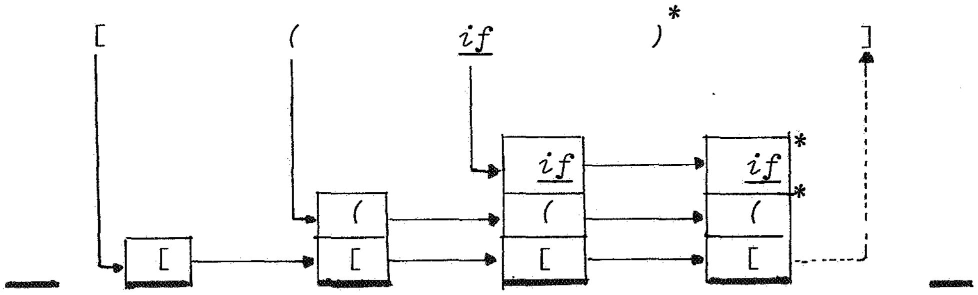
A formatter-symbol which cannot be classified as opening or closing brace and which matches some cell of the stack, is treated as a closing brace; moreover, if it does not match the top, it is treated also as an opening brace. However, in that case, it may not be disregarded in the treatment as a closing brace, since that would give in fact the same interpretation as that in which it is eventually deleted as an opening brace.

3.1.2. Restrictions on the tree

Even now, the algorithm is rather impractical: it has the tendency to generate a large number of inadmissible interpretations (where the definition of "admissible" is the same as that given in 2.2., with the understanding that two transitions are the same if they have the same effect on the top of the stack). For, to each locally admissible interpretation in which some pair of braces match each other, there also exists a locally admissible interpretation in which both braces are marked, i.e., disregarded. In an example:



This interpretation is inadmissible. Clearly, if a closing brace matches the top of the stack, then it may not be disregarded, unless that top will be matched away later on. But even in the last case, the interpretation found cannot be better than the one obtained by honouring the present closing brace. Even if the closing brace does not match the top cell, but some cell lower down the stack, it is dangerous to disregard it. For, if all of the cells down to the one containing the matching opening brace would come to be deleted, an inadmissible interpretation would result too:



Therefore, in such a case, it will be required that at least one of the opening braces on top of the matching one will be matched away later on, i.e., a transition deleting the last of these braces, if none has been matched away yet, will not be admitted. Note that the requirement that the top cell is matched away whenever possible, is a special case of this general requirement.

The tree, grown from left to right, does not fork at opening braces, but may fork at closing braces, where out of four possible types of branches only two are actually used, each consisting of at most one branch:

(i) the closing brace matches away the topmost matching opening brace (and the cells on top of that one, if any, are deleted), and

(ii) the closing brace is disregarded, but at least one of the opening braces on top of the topmost matching opening brace, if any, will be matched away later on.

The two types which are not used, and which may contain several branches, are:

(iii) the closing brace matches away an opening brace below the topmost matching one and deletes all cells on top, and,

(iv) the closing brace is disregarded, and all of the opening braces on top of the topmost matching opening brace will eventually be deleted.

We shall now show that the interpretations obtained in this way are admissible. For, suppose that some interpretation I1 is ruled out by another interpretation I2. In the "extended" tree, obtained by admitting the types (iii) and (iv) also, I1 and I2 are paths from the root of the tree, and so must have a last common node, from which they diverge, i.e., take different branches. Since I1 has been obtained from the original, not extended tree, it must take either a branch of type (i) or one of type (ii). I2 may take a branch of either of the types (i), (ii), (iii) or (iv), but of a type different from that of I1, since (i) and (ii) consist of one branch only and I2 takes a different branch. For each combination we shall derive a contradiction. It is sufficient to show that I2 marks some brace (disregards a closing brace or deletes an opening brace) accepted by I1.

I1	I2	
(i)	(iii)	The opening brace matched away by I1 is deleted by I2.
(i)	(ii) or (iv)	The closing brace honoured by I1 is disregarded by I2.
(ii)	(i), (iii) or (iv)	I1 matches away at least one of the opening braces on top of the topmost matching opening brace later on, whereas I2 deletes all of these.

3.1.3. Further pruning

Even the thus restricted tree, growing at most two branches and then only at closing braces that cannot match away directly an opening brace, may eventually grow so many branches that pruning is necessary. It is obviously not possible to give an absolute guarantee that at that stage the partial interpretation that would give rise to the optimal total interpretation will be retained, but a satisfying solution is obtained by using the following estimate (in fact, lower bound) of the final number of unmatched braces: the number of marked braces thus far, augmented by the absolute value of the difference between the number of cells in the stack, and the net number of closing braces yet to be encountered (where the "net number" is the number of closing braces in the remainder of the brace skeleton minus the number of opening braces). In order to be able to obtain this estimate, it is necessary to have scanned beforehand all braces.

As to the actual implementation of this algorithm, it may be remarked that rather than having complete stacks at the nodes of the tree, it is possible to represent the stacks as the paths in a tree themselves, where the nodes contain one cell. Similarly, the information about marked braces can be kept in a tree which only grows at points where a brace is marked. In this way, the original tree disappears completely, but for a list, corresponding to the nodes of the partial interpretations retained at some stage, containing entries to the other two trees.

4. Ways of correcting the skeleton

Consider the state switcher skeleton from 2.3., in which two state switchers are marked:

¢ ¢ ¢* # ¢ # ¢* .

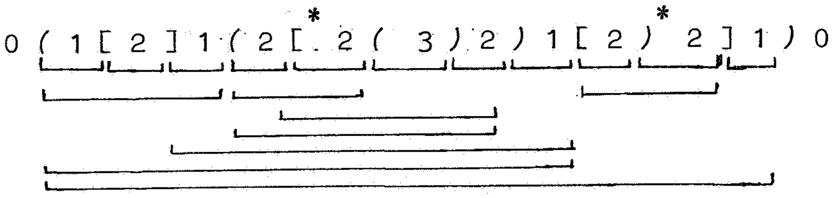
This skeleton can be made correct by deleting the marked state switchers. We have chosen another possibility, that of doubling the marked state switchers, and this state switcher skeleton will be repaired thus:

¢ ¢ ¢¢ # ¢ # ¢¢ .

In the case of the braces, however, some additional actions are possible. Suppose that in the brace skeleton the unmatched braces are marked. Now we can assign level numbers to each of the segments between the braces in the following way: the initial level number is 0; to each segment, as encountered from left to right, the current level number is assigned, and, moreover, if the brace following that segment is an unmarked opening (closing) brace, then the current level number is increased (decreased) by one; the final level number is zero again. The following example shows the level numbers that would be assigned to the segments between the braces of a skeleton:

$$0 \; (\; 1 \; [\; 2 \;] \; 1 \; (\; 2 \; [^{*} \; 2 \; (\; 3 \;) \; 2 \;) \; 1 \; [\; 2 \;)^{*} \; 2 \;] \; 1 \;) \; 0 \; .$$

Furthermore, we define the notion "connected" for two braces in the following way: two braces are connected if the level number, say α, of the segment immediately following the first brace is equal to the level number of the segment immediately preceding the second brace, and nowhere in between those two braces the level number drops below α. For example, in the brace skeleton given above, the connections may be depicted thus:



Now, if in a given brace skeleton a connected series of opening braces occurs, i.e., a series in which each brace is connected to the next, and one of these braces is marked, it is the first one. (This follows directly from our algorithm for the treatment of braces.) In such a case, each of the other opening braces of the connected series is an acceptable alternative for being marked. Therefore, we may look for a better candidate to be marked. Consider the connected series of closing braces, the first one of which is connected with the last opening brace of the connected series of opening braces of which the first one is marked. If we assume that the probability that the missing closing brace should have stood in one of the segments of the connected series of closing braces is equally high for each of the segments, the best place to insert the closing brace is somewhere in a maximal sequence of adjacent closing braces (in the source text) in the connected series of closing braces. If such a maximal sequence exists and has a length which is greater than one, a closing brace is inserted somewhere in this maximal sequence. Consider for example the following brace skeleton, in which one opening brace is marked:

$$\underline{\mathit{begin}}\ \overset{*}{(}\ (\ (\ [\ ]\ (\ (\ [\ ]\ \underbrace{)\ \ )\ \ \ )}_{\text{adjacent}}\ [\ ]\ )\ \underline{\mathit{end}}$$

After the above sketched repairing action, the brace skeleton will be:

$$\underline{\mathit{begin}}\ (\ (\ (\ [\ ]\ (\ (\ [\ ]\ \underbrace{)\ )\ )\ \ )}_{\text{adjacent}}\ [\ ]\ )\ \underline{\mathit{end}}$$

This action is performed in the cases where the marked opening brace is an open-symbol, sub-symbol, begin-symbol or if-symbol. The closing braces close-symbol, bus-symbol, end-symbol and fi-symbol are treated in a similar way.

Another possibility to correct the brace skeleton occurs when, e.g., two braces (and] are both marked, and also connected. In this case, the bus-symbol is changed into a close-symbol. As a last step, two marked and connected formatter-symbols are matched.

4.1. An example

The program given below is example 11.12 from the Report, in which the following errors have been made:

in line 29 3 open-symbols and 2 close-symbols have been inserted;
in line 39 the if-symbol has been omitted;
in line 58 the close-symbol has been replaced by a bus-symbol.

```
'BEGIN' 'MODE' 'RA' = 'REF' 'AUTH', 'RB' = 'REF' 'BOOK',
        'STRUCT' 'AUTH' = ('STRING' NAME, 'RA' NEXT, 'RB' BOOK),
                 'BOOK' = ('STRING' TITLE, 'RB' NEXT);
        'RA' AUTH,FIRST AUTH:= 'NIL',LAST AUTH; 'RB' BOOK;
        'STRING' NAME,TITLE; 'INT' I; 'FILE' INPUT,OUTPUT;
        OPEN(INPUT,,REMOTE IN); OPEN(OUTPUT,,REMOTE OUT);
        OUTF(OUTPUT,$P
               "TO ENTER A NEW AUTHER, TYPE ""AUTHOR"", A SPACE, AND HIS
                NAME."L
               "TO ENTER A NEW BOOK, TYPE ""BOOK"", A SPACE, THE NAME OF THE
                AUTHOR, A NEW LINE, AND THE TITLE."L
              "FOR A LISTING OF THE BOOKS BY AN AUTHOR, TYPE ""LIST"", A SPACE,
                AND HIS NAME."L
              "TO FIND THE AUTHOR OF A BOOK, TYPE ""FIND"", A NEW LINE, AND THE
                TITLE."L
               "TO.END,.TYPE.""END"""AL$,".");
        'PROC' UPDATE =
          : 'IF' ('RA': FIRST AUTH):=: 'NIL'
          'THEN' AUTH:= FIRST AUTH:= LAST AUTH:= 'AUTH':= (NAME,0,0)
          'ELSE' AUTH:= FIRST AUTH; 'WHILE' ('RA': AUTH):≠: 'NIL' 'DO'
                    (NAME = NAME 'OF' AUTH|KNOWN|AUTH:= NEXT 'OF' AUTH);
                    LAST AUTH:= NEXT 'OF' LAST AUTH:= AUTH:= 'AUTH':=
                    (NAME,0,0); KNOWN: 'SKIP'
                'FI' # END DECLARATION PRELUDE SEQUENCE #;
CLIENT: INF (INPUT,$C("AUTHOR","BOOK","LIST","FIND","END",""),X30AL,
                       80AL$,I);
        'CASE' I 'IN' AUTHOR,PUBL,LIST,FIND,END,ERROR 'ESAC';
AUTHOR: IN (INPUT,NAME); UPDATE; CLIENT;
PUBL:((IN(('NPUT, (NAME,TITLE))))

INCORRECT TRANSITION:BRACE ADDED

                                 ; UPDATE;
                 'IF' ('RB': BOOK 'OF' AUTH):=: 'NIL'
               'THEN' BOOK 'OF' AUTH:= 'BOOK':= (TITLE,0)
               'ELSE' BOOK:= BOOK 'OF' AUTH; 'WHILE' ('RB': NEXT 'OF' BOOK) :≠:
                  'NIL' 'DO'
                    (TITLE = TITLE 'OF' BOOK|CLIENT|BOOK:= NEXT 'OF' BOOK);
                 (TITLE ≠ TITLE 'OF' BOOK|NEXT 'OF' BOOK:= 'BOOK':= (TITLE,0))
                'FI'; CLIENT;
LIST:   IN (INPUT,NAME); UPDATE;
        OUTF(OUTPUT,$P"AUTHOR:."30ALL$,NAME);
                      ('RB': BOOK 'OF' AUTH):=: 'NIL'
        'THEN'

INCORRECT TRANSITION:CORRESPONDING BRACE IS MISSING

               PUT(OUTPUT,"NO PUBLICATIONS")
        'ELSE' 'WHILE' ('RB':BOOK) :≠: 'NIL' 'DO'
                'BEGIN' 'IF' LINENUMBER(OUTPUT) = MAX LINE[REMOTE OUT]
                  'THEN' OUTF(OUTPUT,$41K"CONTINUED ON NEXT PAGE"P
                              "AUTHOR: "30A41K"CONTINUED"LL$,NAME)
                  'FI'; OUTF(OUTPUT,$80AL$,TITLE 'OF' BOOK);
                BOOK:= NEXT 'OF' BOOK
                'END'
        'FI'; CLIENT;
FIND:   IN (INPUT, ('LOC' 'STRING',TITLE)); AUTH:= FIRST AUTH;

        'WHILE' ('RA': AUTH) :≠: 'NIL' 'DO'
          'BEGIN' BOOK:= BOOK 'OF' AUTH;
                'WHILE'('RB': BOOK) :≠: 'NIL' 'DO'
                    'IF' TITLE = TITLE 'OF' BOOK
                'THEN' OUTF (OUTPUT,$L"AUTHOR: "30A$,NAME 'OF' AUTH); CLIENT
                'ELSE' BOOK:= NEXT 'OF' BOOK
                  'FI'; AUTH:= NEXT 'OF' AUTH
        'END';
        OUTF(OUTPUT,$L"UNKNOWN"L$,]

INCORRECT TRANSITION:BRACE CHANGED INTO: )

                                   ; CLIENT;
END:    PUT(OUTPUT,(NEWPAGE,"SIGNED OFF",CLOSE));
        CLOSE(INPUT).
ERROR:  PUT(OUTPUT,(NEW LINE,"MISTAKE, TRY AGAIN."));
        NEW LINE(INPUT); CLIENT
'END'
```