Transformational Decomposition :

A Simple Description of an Algorithm for Transformational Analysis of English Sentences*

Danuta Hiż

Aravind K. Joshi

University of Pennsylvania

ABSTRACT

In this paper, we will present a rather simplified description of an algorithm for transformational analysis (decomposition) of English sentences. Our purpose here is not to discuss the transformational theory, the full details of the theoretical formulations of the algorithm, or of the grammar. Rather, we will present a set of examples of the decomposition and some discussion of them with the hope that it will give enough insight into the capability of the algorithm and indicate to some extent the power of transformational analysis.

*This work was carried out in the Transformations and Discourse Analysis Project, University of Pennsylvania, sponsored by the National Science Foundation.

1.0 Here, we will present a rather simplified description of an algorithm for transformational analysis (decomposition) of English sentences. Our purpose here is not to discuss the transformational theory, the full details of the theoretical formulations of the algorithm, or of the grammar*. Rather, we will present a set of examples of the decomposition and some discussion of them with the hope that it will give enough insight into the capability of the algorithm and indicate to some extent the power of transformational analysis.

1.1 Transformations are certain relations among sets of sentences and in particular, it is possible to relate a given sentence to a set of elementary sentences (kernel sentences) by means of transformations. The kernel sentence forms (for English) are defined as the string of class marks N t V followed by one of the kernel object strings: φ, N, NN, NPN, ND, PN, D, A (N: Noun; t: tense/aux; V: verb; P: preposition; D: adverb; A: adjective; φ : zero). Thus John bought a book; Mary will come etc. are kernel sentences. Each transformation is characterized by certain permutations, deletions or additions of specific class marks or constants. In the resultant of a transformation one may look for subsequences which remain invariant even when the resultant is subjected to further transformations. The basic features of the algorithm are

a) stating the various invariant sequences and

b) formulating 1) a grammar of such invariant sequences, 2) a corresponding recognition procedure, and 3) a systematic procedure for computing the kernel sentences as well as other kernel-like sentences and the corresponding transformational history.

It should be emphasized that it is not assumed and also not implied in the algorithm that any kind of prior analysis (either string analysis or constituent analysis) is required as a prerequisite for the present algorithm.

---

*Such a detailed description will appear later elsewhere.

1.2 Transformations are initially defined on kernel sentence forms. However, they work on certain other sentence forms which are not kernel sentence forms. Thus a transformation is completely defined by first defining it on a suitable kernel sentence form(s) and then extending the domain of the transformation to other sentence forms. This extension which contains infinitely many sentence forms can be represented by first listing a finite number of sentence forms in the extension and all the remaining sentence forms in the extension are obtained by certain recursive rules (see the i-lists in 1.3).

1.3 A unary transformation transforms one sentence form into another sentence form and a binary transformation transforms a pair of sentence forms into another sentence form. Each unary transformation defined on a sentence form may be represented by a sequence of class marks constituting another sentence form. Most binary transformations can be defined as interruptions of certain unary transformation sequences at stated positions by certain other sequences of class marks. These interrupting sequences are not sentence forms but are deformations of sentence forms corresponding to the second sentence form of the binary transformation. For example, John was detained by the old woman decomposes into woman detained John and woman t be old with a passive transformation on the first kernel and a binary transformation on the sentence John was detained by the woman and the kernel sentence woman t be old.[1] The sentence form corresponding to the passive transformation, N t be en V by N is then interrupted by the sequence AN before the last symbol. AN is a deformation of the kernel sentence form N t be A which is the second sentence form of the binary transformation. The resulting sentence form is thus N t be en V by A N. In the resulting sentence form the shared symbol N appears only once. Such a symbol which two transformation sequences share (or on which they overlap)

[1]We ignore here the article the for simplicity.

will be called a residue of one sequence with respect to another.[2] In addition to the transformation sequences which are sentence forms, and the interrupting sequences (deformed sentence forms) which correspond to most binary transformations, there is yet another type of interrupting sequences (again deformed sentence forms) which correspond to nominalizations. For example consider: the book was written by Brown and John's travel to Italy was described by Mary. In the second sentence, the kernel sentence John travelled to Italy is mapped onto the object of Mary described before the resultant undergoes the same passive which acted on Brown wrote the book giving the first sentence. N's nV P N (John's travel to Italy) is a nominalization which appears in many different transformations and carries in them the associated kernel into one of the positions which could be occupied by a noun. For each transformation sequence in each intersymbol position we list all interrupting sequences (including both the second and the third kind of sequences as discussed above). Of course, the interrupting sequences have their own interrupting sequences, etc. These intersymbol interrupting lists will be called i-lists.

2. A sketch of the algorithm

2.0 As stated in 1, in order to define the set of all transforms we need a set of sequences of class marks (or class mark-like symbols) which has 3 subsets.

1. Sequences each of which corresponds to a sentence form (e.g. the passive sequence N t be en V by N);
2. Sequences each of which represents a deformed kernel-form and is not a sentence form, but when inserted between specified neighboring symbols of a sequence of the first set, preserves the character of the sentence form (e.g. AN, en V N);
3. Sequences each of which represents a deformed kernel-form and is

[2] The concept of the residue can be extended to shared sequences as well as sequences which replace a given symbol in another sequence. The term carrier is used in this context. This device has been extensively used in this algorithm.

not a sentence form, but, when substituted for a symbol in a sequence (of set 1 or 2 or 3), preserves the character of that sequence (e.g. er V or Ω, n A of N).

There are also rules for inserting sequences from the second set into other sequences or into sequences of the third set, without changing the character of either.

All insertion or replacement rules are stated in the interruption lists appearing between every pair of adjacent symbols of each sequence.

Most of the sequences in the first set represent unary transformations of kernel forms. Many are extended (often by permitting the replacement of certain symbols with selected sequences from the third set) to include analogous unary transforms of kernel-like forms.

The second set of sequences, together with the rules of their insertion in the sequences for unary transformations, account for most of the binary transformations. Other binary transformations are represented by replacement in pairs of class marks in unary transformation sequences by members of the third set, most of which consist of nominalizations.

An arbitrarily long English sentence form can be seen as composed of a finite number of such sequences recursively embedded in one another.

2.1 Corresponding to the above three subsets of sequences and their mutual embedding rules, we recognize three sets of strings. Each string is a program for comparing one of the sequences with a portion of the analyzed sentence form of the data. The program is equipped to permit interruption by other such programs according to the i-lists of the sequence. Each string, when entirely matched by a segment of data, replaces that segment with the carrier of the string. The carrier is sometimes null. In strings from the second set it is usually the residue of the binary insert (e.g. the center symbol of a noun phrase: N of AN, of ing V N,etc.). In strings from the third set the carrier is a class-mark-like symbol which, by replacing a classmark in a form derived from a kernel form, extends it to one similarly derived from a kernel-like form. Let the carrier be $\tilde{N}$[nV] for a noun phrase built around an nV. The extended passive form: N[or $\tilde{N}$[nV]] t be en V by N represents the form of the sentence John's travel to Italy was described by Mary as soon as the carrier of the string replaces in the data the nominal segment John's travel to Italy. The carrier from all strings in the first set is S, a symbol of a well-formed sentence.

The program of each string, whose sequence is a deformed (or transformed) kernel or kernel-like form, reconstructs that form for decomposition and attaches to it a label descriptive of the deformation (or transformation). The result of a decomposition is a set of kernel or kernel-like sentences with labels. Some of the kernel sentences are incomplete and have blanks in them because a transformation may delete elements. Some kernel-like sentences may contain, instead of a word, a class-mark-like symbol (e.g. $\tilde{N}$) with a reference to a previous component of the decomposition. If that previous component is a kernel sentence (with or without blanks), then the label (describing the deformation) with the kernel-like form (containing the reference) with its label, together constitute a description of the transformation undergone by the component kernel sentence. If the previous component itself is a kernel-like sentence with a reference in turn to another component, both kernel-like sentences and all three labels constitute the description of the transformation undergone by the component kernel sentence ultimately referred to, etc.

If the symbol x appears, instead of a word, in a kernel or kernel-like sentence, it replaces a regular noun there. It is introduced in the sentence as a carrier from a nominalization such as a teacher of Latin, the driving instructor, etc. The same x must appear in two or more sentences of the decomposition (one where the nominal stands for a noun, and one in the sentence of which the nominalization is a deformation, e.g. x - teach Latin). Which x's require identical substitutions is discoverable, because each x has in a sharp bracket (< >) the names of every previous line in which the same x appeared. Often no actual substitution is possible and the x serves only to identify, with each other, two or more blanks in different components. The substitution of the noun replacing N for x in lines a, b, . . . d is implied when one kernel-like component has the form N t be x <a, b . . . d>.

2.2 The three sets of strings (programs) constitute the major portion of grammatical material in the algorithm. Another body of such material is the dictionary.

The dictionary associates to each English word a symbol representing the word's grammatical class, together with markers of certain additional characteristics the word may reveal by restricting its environment in the sentence. Some words may occur in more than one role and have therefore several equivalents in the dictionary. (e.g. the word labor should be given four different class marks: present tense V, V(untensed verb), nV (nominalization designating the activity of laboring), er V (nominalization designating the actor(s), possibly laborers in aggregate)).

The dictionary for Transformational Grammar must carry far more details than is needed for the String Analysis alone. Thus for example the transformational analysis must be able to discover in John's sleep not only a noun phrase, but also the incomplete kernel sentence John-sleep φ which underlies each transformation containing such a noun phrase. Hence the class marks: nV (sleep), ing V (sharing) nA (bravery), erV (teacher), eeV (employee), nN (brotherhood), aV (helpful) and several others.

A V-entry in the String Analysis dictionary contains information about the kind of objects required by the verb V. An nV may require objects different from its V and this must be indicated (e.g. they attacked the enemy vs. they made an attack on the enemy).

Noun phrases like nV, ingV, etc. can occur in place of a sentence object or a subject of a sentence but only when it is organized around a verb requiring such subjects or objects, and such verbs are marked accordingly in the dictionary.

The subject and object restrictions for a verb or a verb-related word are recorded in pairs, because they are not mutually independent. (σ is the label for a subject (Σ) requirement; $\underline{\omega}$ for an object (Ω) requirement of a tensed or untensed verb and some ingV occurrences; ω_{nV} labels an object requirement of nV-nominalization, ω_{ingV} those of an ing V-nominalization, etc. When needed, $\underline{\omega}_1$ is distinguished from $\underline{\omega}_2$ (which usually is the same as the corresponding $\underline{\omega}$) to mark the form assumed by the object when it precedes the verb or verb related word (compare for instance house construction with construction of house where ω_{nV1} (the same as ω) is N, while ω_{nV2} (the same as ω_{nV}) is P [of] N).)

The analysis is preceded by a replacement of the words in the sentence by corresponding entries in the dictionary.

2.3 The process of analyzing a sentence begins in postulating (in turn) all those strings in the grammar which may occur at the beginning of a sentence (and whose initial symbol is the same as the first symbol in the data). (See i_1 of #30). Each verified postulate forces other postulates as its consequences, until the terminal period of the sentence is found which is consistent with a hypothesis. It is quite likely that an analysis will produce more than one correct reading of a sentence, because structural ambiguity is even more frequent in transformational grammar than it is in the mere string analysis.

3. Examples of decomposition

Four examples of decomposition obtained by the algorithm follow. These examples are intended to exhibit the power of the algorithm.

It is possible, without changing the algorithm, to increase the power and depth of the analysis by incorporating more details about transformations as they become available by adding either new transformation sequences or adding new classes and new co-occurence restrictions in the dictionary or both.

Among the various issues which are now receiving further attention, some are as follows: a) a better characterization of nF-nouns and the underlying kernel sentences in terms of which the modifiers can be explained (e.g. school principal (example 3), French teacher; British queen, etc); b) the relation of classifier nouns to each other and their kernel positions with respect to their modifiers (e.g. organic chemistry, helpful trip, friendly letter, etc.); c) precise relation of constants (e.g. his, both in example 4) or classifier nouns with a definite article to other nouns or phrases for which they are a replacement.

Examples: The first column lists the kernel sentences or kernel-like sentences (or intermediate resultants). The second column gives the rest of the transformational history. Here the names as stated are partial in the sense that the corresponding strings do not always correspond to complete transformatial sequences as discussed previously.

1.

Text:	The	fact	that	John	is	a	stranger	makes	
	T	N_w	that	N	pres.be[3]	T	N	present V[3][1]	
	his	life	here	unbearable.					
	R's	nV	D	aV					

[1] 3 indicates here 3rd person.

Decomposition:

	Kernel or Kernel-like sentences	transformation (partial names)	carrier
1.	John pres.be stranger (a)	container[2] noun: N_w that S	N_w ⟨1⟩
2.	He - live here;	$\tilde{N}$-nominaliztion; Σ's nVΩ	$\tilde{N}$ ⟨2⟩
3.	- cannot bear $\tilde{N}$⟨2⟩	adjectivization : aV	$\tilde{A}$ ⟨3⟩
4.	N_w ⟨1⟩ pres. make $\tilde{N}$ ⟨2⟩ $\tilde{A}$ ⟨3⟩	container: N V_w NA	S

2.

Text:	Our	algebra	teacher	was	requested	by	the school
	R's	N	erV	past be [3]	enV	by	T N

principal	to	interview	a	woman	candidate	from	Swarthmore.
nF	to	V	T	N	N	P	N

Decomposition:

Kernel or Kernel-like sentences	transformations (partial names)	carrier
first reading:		
1. x - teach algebra	x-nominalization: ΩerV	x ⟨1⟩
2. We- have x⟨1⟩	left modified noun: N's N	x ⟨1,2⟩
3. x - heads[3] school	x-nominalization: NnF	x ⟨3⟩
4. woman- V_{app} P_{app} candidate (a) (V_{app} = be; P_{app} = φ)	compound noun: N_1N_2	candidate⟨4⟩
5. candidate-be from Swarthmore	noun, right modified: N_1 P N_2	candidate ⟨4,5⟩
6. x ⟨1,2⟩ - interview candidate ⟨4,5⟩ }	passive of container: N t V_w N infinitive	S
7. x ⟨3⟩ past request x ⟨1,2⟩ ⟨6⟩ }		

---

[2]Roughly, container forms are sentence forms in which 1) there is a verb (V_w) requiring a sentential subject or a sentential object or both or 2) there is a noun (N_w) or adjective (A_w) requiring sentential complements.

[3]heads is a $V_{appropriate}$ for nF principal as found in dictionary.

Kernel or Kernel-like sentences	transformations (partial names)		carrier
second reading:			
1. x – teach us algebra	x-nominalization:	Ω erV	x ⟨1⟩
2. x – head school	x-nominalization:	NnF	x ⟨2⟩
3. woman – V_{app} P_{app} candidate (a)	compound noun:	N_1N_2	candidate ⟨3⟩
4. candidate-be from Swarthmore	noun, right modified: N_1PN_2		candidate ⟨3,4⟩
5. x⟨1⟩ – interview candidate ⟨3,4⟩	passive of container: NtV_wN		S
6. x⟨2⟩ past request x ⟨1⟩⟨5⟩	infinitive		

3.

Text:	Accident	insurance	of	an	employee	by	his	employer
	N	nV	P	T	eeV	P	R's	erV

protects		both.
present	V: [3]	Q

Decomposition:

Kernel or Kernel-like sentences	transformation (partial names)	carrier
first reading:		
1. – – employ x	x-nominalization: eeV	x ⟨1⟩
2. x – employ him	x-nominalization: erV	x ⟨2⟩
3. x ⟨2⟩– insure x ⟨1⟩(an) P_{app} accident	$\tilde{N}$-nominalization: nV	$\tilde{N}$[nV+Ω +Σ] ⟨3⟩
4. $\tilde{N}$ ⟨3⟩ present protect both	container: $\tilde{N}$ t V_w N*	S
second reading:		
1. – – employ x	x-nominalization: eeV	x ⟨1⟩
2. x – employ –	x-nominalization: erV	x ⟨2⟩
3. he – have x ⟨2⟩	left modified noun: $N'sN_2$	x ⟨2,3⟩
4. x ⟨2⟩ – insure x ⟨1⟩(in) P_{app} accident	$\tilde{N}$-nominalization: nV	$\tilde{N}$[nV+Ω +Σ] ⟨3⟩
5. N ⟨4⟩ present protect both	container: N t V_w N	S

Note: The analysis would reach even deeper if the words his and both were treated as reference words leading to a substitution, e.g. of x ⟨1⟩ for he, x ⟨1⟩ and x ⟨2⟩ for both.

4.

Text:	Crop	sharing	between	the	tenant	and	the	land	owner
	N	ing V	P	T	N	and	T	N	erV

* Q may replace N.

	is	an	economic	arrangement	unsatisfactory	to
present be [3]		T	aN	nV	aV	P

organized labor.
enV erV

Decomposition:

Kernel or Kernel-like sentences	transformations (partial names)	carrier
1. x - own land	x-nominalization: erV	$\tilde{x}$ $\langle 1 \rangle$
2. tenant (the) and x $\langle 1 \rangle$(the)-share crop	$\widetilde{N}$-nominalization: ingV	$\widetilde{N}$[ingV+ $\Sigma+\Omega$] $\langle 2 \rangle$
3. - - arrange -; P_{app} economy	$\widetilde{N}$-nominalization: nV	$\widetilde{N}$[nv] $\langle 3 \rangle$
4. x - labor -	x-nominalization: erV	x $\langle 4 \rangle$
5. - - organize x $\langle 4 \rangle$	left modified noun: enVN	x$\langle 4,5 \rangle$
6. $\widetilde{N}$ $\langle 3 \rangle$ -not satisfy x $\langle 4,5 \rangle$	$\widetilde{N}$ right modified: $\widetilde{N}$ aV	$\widetilde{N}$ $\langle 3,6 \rangle$
7. $\widetilde{N}$ $\langle 2 \rangle$ present be $\widetilde{N}$ $\langle 3,6 \rangle$ (an)	container: $\widetilde{N}$t be $\widetilde{N}$	S

4. An illustration of the procedure

Example 5 John is a good story teller

This example illustrates the process of analysis in some detail. Because of space limitations for this paper a rather simple structure had to be chosen for this purpose. A short dictionary of the words in the sentence has been prepared and also a small set of grammar strings in provided for this illustration. Both were greatly simplified so that rich grammatical material will not obscure the demonstration of the choice of hypotheses, their verification or rejection, the use of the carrier, changes of levels in analysis and the exploration of alternative readings.

The analysis always begins with the string #30 postulated. A decomposition ends, when the program associated with this string is finished. All possible sentence beginnings are included in i_1 of #30. After the end of #30 alternative decompositions are sought.

When a new string is postulated on the basis of an i-list of another string, the verification of the new string takes place in the next level of a push-down memory, so that the state of computation of the suspended string is not affected.

Whenever two or more alternative paths open up for the analysis, each must be pursued to a successful completion or until failure occurs. (The analysis must produce every possible decomposition of a structurally ambiguous sentence). In our analysis, different paths are pursued serially. Every time an inspection of i-lists allows more than one hypothesis, one is chosen, while a list of the remaining ones together with all relevant positions of the memory goes on top of another push-down storage. The contents of that storage is examined after the end of the chosen path. The analysis ends after all possible paths have been explored and this storage is empty. In the example of analysis given here, we mark bypassed open branches by asterisks on the left margin and their resumption by similar asterisks encircled.

Dictionary used in Example 5.

John - N [proper, human, singular]
is - present be [3[1]; σ: N[or x]; ω: N/A/PN/D.
σ: $\tilde{N}$[nV/ingV]; ω: A_w, N_w, $\tilde{N}$[nV/ingV] etc.]
a - T[a]
good - A [A-ly = well]
story - N
teller - erV [σ : human, count; $\omega = \omega_{erV1}$: N/$\tilde{N}$[nV]/φ; ω_{erV2};
PN[or x or $\tilde{N}$: nV]/φ]

Grammar Strings used in example 5.

Nominal strings (each gives a noun-like carrier):

1. T[the/a/an] i_1 N[or x; or $\tilde{N}$: nV/ingV/nA/nN] i_2
i_1 2,3,4,5
i_2 -
name: -
kernel: -
carrier: N[the article] (as matched)

2. A i N[or x]
i_1 2,4,5,
i_2 -
name: left modified noun: AN
kernel: N - be A (N,A as matched from data)
carrier: N < address of kernel > (N " " " ")

[1]Designates third person.

3. A ^{i_1} $\tilde{N}$[nV/ingV; or x]^{i_2}

i_1 3,

i_2 -

name: left modified nominal: A $\tilde{N}$ [or x]
addition to kernel of $\tilde{N}$: ; A-ly (A, as matched)
carrier: $\tilde{N}$ ($\tilde{N}$ as matched from data)

4. erV ^{i_1}

i_1 -

name: x-nominalization
kernel: x-V-
carrier: x [subclasses required from subject of V] ⟨address of kernel⟩

5. N[object] ^{i_1} erV ^{i_2}

i_1 -

i_2 -

name: x-nominalization
kernel: x- V N (N,V as matched from data).
carrier: x [subclasses required from subject of V] ⟨address of kernel⟩

6. nV ^{i_1} Ω_{nv} ^{i_2}

i_1 -

i_2 -

name: $\tilde{N}$-nominalization: nV[+Ω]
kernel: $\tilde{-}$ - V Ω_{nV} (as matched from data)
carrier: $\tilde{N}$[nV] ⟨address of kernel⟩

Object Strings:

10 - N[or x]^{i_1}

i_1 -

name - object
contribution to kernel in carrier
carrier: Ω[N] (N as matched in data)

11 - N[or x] ^{i_1} P ^{i_2} N[or x] ^{i_3}

i_1 -

i_2 1,2,4,5

i_3 -

name - object
contribution to kernel in carrier
carrier: Ω[N P N] (N P N as matched in data)

Sentence Strings

i_1 i_2 i_3 i_4

20 - N t V Ω

i_1 -

i_2 -

i_3 - 10,11,1,2,3,4,5,6

i_4 -

name: identity of kernel form: N t V Ω
kernel: N t V Ω (as found in data)
carrier: S < address of kernel>

i_1 i_2 i_3 i_4

21 - $\tilde{N}$ t be $\tilde{N}$

i_1 -

i_2 -

i_3 - 1,3,6

i_4 -

name: containing "be" ; $\tilde{N}$ is $\tilde{N}$
kernel: $\tilde{N}$ t be $\tilde{N}$ (as matched from data)
carrier: S < address of kernel>

Monitor String

i_1 i_2 i_3

30 - . S .

i_1 1,2,3,4,5,6,20,21

i_2 -

i_3 -

<u>Illustration of the process of analysis:</u>

Data: . N[John] pres.V[3,be] T[a] A[good] N[story] erV[teller] .
#30: . S . (level 1)
$\dot{.} = \dot{.}$
S ≠ N
i_1 of 30 allows the following strings beginning with <u>N</u> to interupt 30 here: 5,20. Try 20, mark <u>*</u> for the branch opening with 5 on level 2.

* Data: N[John] pres V[3,be] T[a] A[good] N[story] er V[teller]
#20 : N tV Ω (level 2)
N = N[John]
t = present
V = V <u>be</u> accepts <u>John</u> as subject. For a human subject, the object cannot be ≠ 6(in this simplified grammar). The verb <u>be</u>

rejects object form of #11.

$\Omega \neq T$ Among the remaining strings of i_3 of 20 (1,2,3,4,5,10) only 1 starts by T.

Data: T[a] A[good] N[story] erV[teller]. (level 3)
#1 : T N
T = T
N ≠ A
i_1 of 1 has 2,3 beginning with A. Try 2, mark *** for bypassed 3.

*** Data: A[good] N[story] er V[teller]. (level 4).
#2 : A N
A = A
** N = N note: i_1 of 2 has string 5 beginning with N.
Mark ** for the bypassed branch.
end 2. kernel 1: story-be good. Resume 1.

Data: N[story] <1> erV[teller]. (level 3)
continue #1

*** N = N note: i_1 of 1 has string 5 beginning with N. Mark open branch ***. end 1. Resume 20.

Data: N[story] < 1 > (a) er V[teller]. (level 2)
continue #20
$\Omega \neq N$
of the strings from i_3 allowed by object requirement of the verb be, 5 and 10 begin with N. Try 10, mark **** for bypassed 5.

**** Data: N[story] <1> (a) erV[teller]. (level 3)
#10: N
N = N
end 10. Resume 20.

Data: Ω [N[Story] <1> (a)] erV[teller]. (level 2)
continue #20.
$\Omega = \Omega$ [N[story] <1> (a)]
End 20. Kernel 2: John pres. be story <1> (a). Resume 30.

Data: S <2> erV[teller]. (level 1)
continue #30
S = S
· ≠ er
There is no string in i_2 of 30 which begins with er. Resume the nearest open branch: #5 at level 3. Erase mark **** (kernel 2 is also erased)

(****) Data: N[story] <1> (a) erV[teller]. (level 3)
#5 : N erV
N = N
er = er
V = V story is a proper object for teller

```
    End 5.  Kernel 2:  x - tell story <1>(a).  Resume 20.

Data:  x <2>.     (level 2)
continue #20
Ω ≠ x
The only string beginning with x among those of i3 allowed as
object of be is 10

    Data:  x < 2>.    (level 3)
    #10:   N[or x]
    N = x
    end of 10.  Resume #20.

Data:  Ω[x <2>]
continue #20.
Ω = Ω [x[human, ct., singular] < 2>]. However, the verb be with
       a count-noun subject requires from a noun object an ar-
       ticle or an article-replacer.  This lacking, the current
       branch fails, the branch marked *** is reopened with #5
       on level 4.  (Kernel 2 of the failing branch is erased.)
       Erase ***.

  (***) Data:  N[story] erV[teller].    (level 4)
        #5  :  N  erV
        N = N
        er = er
        V = V story is appropriate object for teller.
        End 5.  Kernel 2: x - tell story < 1>.  Resume 1.

    Data:  x < 2>.        (level 3)
    continue #1
    N = x
    end 1.  Resume 20.

Data:  x  < 2>(a) .     (level 2)
continue 20
Ω ≠ x
only one string beginning by x can interrupt here; it is 10.

    Data:  x < 2>(a).    (Level 3)
    #10 :  N[or x]
    N = x
    end 10.  Resume 20
Data:  Ω[x < 2>(a)]  .     (level 2)
continue #20
Ω = Ω
end 20.  Kernel 3:  John pres. be x < 2>(a).  Resume 30.
Data:   S .              (level 1)
continue #30
S = S
. = .
End 30
Print output:  1. story - be good            (left modified noun)
               2. x - tell story < 1 >        (x-nominalization: ΩerV)
               3. John present be x < 2> (a) (identity of extended NtVN)
```

Are there any branches open? Yes, ** at level 5. #5 will be tried.
Erase **.

(**) – Data: N[story] erV[teller]. (level 5)
#5 : N erV
N = N
er = er
V = V story is appropriate object of teller
end 5. Kernel 1a: x - tell story. Resume 2.

Data: x ⟨1⟩ . (level 4)
continue #2
N = x
End 2. Kernel 2a: x ⟨1⟩ - be good. Resume 1.

Data: x ⟨1,2⟩ . (level 3)
continue #1
N = x
end 1. Resume 20.

Data: x ⟨1,2⟩ (a) . (level 2)
continue #20
$\Omega \neq x$
The only string allowed to interrupt here is 10.

Data: x ⟨1,2⟩ (a) . (level 3)
#10 : N[or x]
N = x
End 10. Resume 20.

Data: Ω[x ⟨1,2⟩ (a)] . (level 2)
continue #20:
Ω = Ω
End 20. Kernel 3a: John pres. be x ⟨1,2⟩ (a). Resume 30.

Data: S ⟨3⟩ . (level 1)
continue 30
S = S
. = .
end 30.

print output:

1. x - tell story (x-nominalization: ΩerV)
2. x ⟨1⟩ - be good (left modifier noun)
3. John pres. be x ⟨1,2⟩ (a) (identity of extended NtVN)

Are there any branches open? Yes, *** at level 4.
(To abbreviate, we will just say that this branch will be very much like the last one, except that, due to the difference between strings 2 and 3, it will give the output:

1. x- tell story; well (x-nominalication: ΩerV; left modified nominal)

2. John pres. be x < 1 > (a) (identity of extended NtVN)

The last open branch, marked * fails immediately.)

References

1. Joshi, A. K., Hiż, D., "String representation of transformations and a decomposition procedure", Part I and Part II, Transformations and Discourse Analysis Project Paper, University of Pennsylvania, Dec. 1965.

2. Joshi, A. K., "Transformational analysis by computer with some applications", Presented at the National Institute of Health Seminar on Computational Linguistics, Bethesda, Oct. 1966. (To be published).