

[From: A.Zampolli, N.Calzolari (eds.) *Computational and mathematical linguistics*. Proceedings of the International Conference on Computational Linguistics, Pisa, 1973, vol.II (Firenze: Olschki, 1980)]

ALAN K. MELBY

## JUNCTION GRAMMAR AND MACHINE ASSISTED TRANSLATION

### 1. *The Junction Grammar Model of Language.*

Junction Grammar is a generative grammar with multiple levels of representation. The deep level is a nonlexical conceptual level, where the meaning of a segment of discourse is represented by a set of intersecting branching trees, each tree consisting of conceptual units (or sememes) and junctions.<sup>1</sup> The junction patterns (whence the name, Junction Grammar) are selected from a finite language-independent pool, and are used to describe the relationships between the sememes.

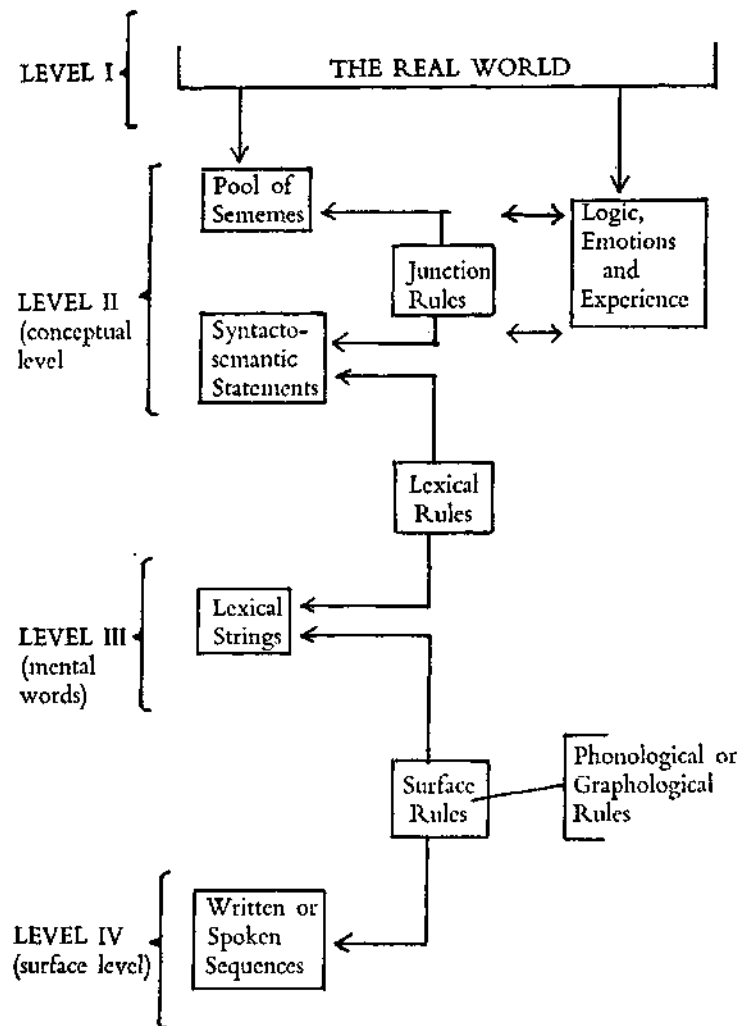
The basic concepts of Junction Grammar were developed at Brigham Young University by Dr. Eldon G. Lytle from 1968 to 1970, but significant refinements to the theory continue to be added. We will first examine the Junction Grammar model of language, then the model of translation suggested by the model of language, and finally an application of the model to mechanical translation or, as we prefer to term it, computer-assisted translation.

The Junction Grammar model distinguishes four levels of representation (see Fig. 1).

Level I is the real world, where objects represent themselves. Level II is the conceptual level at which we postulate that a person has a pool of base sememes from which he draws to form segments of discourse by the application of junction rules. These sememes correspond to a particular meaning of a word, phrase, or part of a word at level IV. At level II, we call a meaningful segment of discourse a "Well-Formed Syntacto-semantic Statement" or WFSS (pronounced /wufəsəs/). A WFSS is a statement about the syntax which joins a set of concepts to form a segment of discourse. This level II syntax is not to be confused with surface syntax, which deals with such matters as word order. Our

---

<sup>1</sup> Other work in the area of conceptual grammar includes that of Roger Schank at Stanford University. See various ARPA reports.



Note: Levels II and III are in the mind.

Fig. 1. *The Junction Grammar Model of Language*

level II sememes are not words, but rather the concepts behind them. Words are not introduced until level III. In fact, our model requires the application of a set of lexical rules to pass from level II to level III. Our lexical rules govern ordering (to order the elements of a WFSS), hiatus (to omit understood elements), matching (to match sememes with lexemes), and agreement (to add inflections). They apply to a WFSS



pair of nodes from which the word *whom* is lexicalized. The level II representation contains no words, but only arbitrarily assigned semantic indices and features interrelated by junction patterns. Of course, the semantic indices are not independent of each other. For example, the concept “boy” is a subset of the concept “human”. This relationship is derived from a person’s experience and logic. We have not represented this facet of level II in Fig. 2. Instead we have used the stopgap approach of assigning the feature < + human > to the index for “boy”.

The inherent relationships between concepts form a kind of semantic syntax.<sup>2</sup> These relationships are important in a model of language, but our level II intersecting trees show another kind of semantic syntax: the temporary junctions formed between sememes to create a segment of discourse.

It is important to note that the order of level II elements does not vary when different word order options are used in the input sentence, but is governed by the form of the junction rules. In level III, a sentence consists of a string of lexemes in the same order as the words of the surface level sentence. In Fig. 3, we represent the lexemes at level III by indicating the key necessary to retrieve the level IV word.

Level III	Level IV
<i>ENG</i> (214.0)	<i>it</i>
<i>ENG</i> (119.0) + splice	<i>surprise</i> +
<i>ENG</i> (0.3)	<i>d</i>
<i>ENG</i> (299.2)	<i>me</i>
<i>ENG</i> (0.9)	<i>that</i>
<i>ENG</i> (0.2)	<i>the</i>
<i>ENG</i> (376.0)	<i>boy</i>
<i>ENG</i> (0.8)	<i>whom</i>
<i>ENG</i> (299.3)	<i>we</i>
<i>ENG</i> (186.2)	<i>saw</i>
<i>ENG</i> (248.2)	<i>caught</i>
<i>ENG</i> (179.0)	<i>many</i>
<i>ENG</i> (235.1)	<i>fish</i>

Fig. 3. *Semantic Indices*

<sup>2</sup> See S. M. LAMB, *Lexicology and semantics*, in A. A. HILL (ed.) *Linguistics Today*, New York-London 1969, pp. 45-46.

This key consists of the semantic index and the lexical agreement displacement. The displacement is a number calculated by language specific lexical agreement rules. Lexemes not derived from any particular semantic index are represented by a zero index and a language-specific displacement.

Now let us briefly examine the set of junction rules used to form level II representations. The junction rules are computable from two basic schema (involving three junction operations) and a list of axioms. The schema are full-junction and interjunction (see Fig. 4).

Full Junction		Interjunction
$X \text{ operation } Y \Rightarrow Z$		$X * Y \text{ (subordinate rule)} \Rightarrow X$
examples:		examples:
$V + N \Rightarrow PV$	(predicate with transitive verb)	$N * N (PV + N/L) \Rightarrow N$ (who relative)
$N * SV \Rightarrow N$	(noun complement)	$N * N (V + N/L) \Rightarrow N$ (whom relative)
$P + N \Rightarrow PP$	(prepositional predicate)	$SV * N (PV + N/L) \Rightarrow SV$ (sentence relative)
$PV + N \Rightarrow SV$	(predication with verbal predicate)	$N * N (A * N/L) \Rightarrow N$ (whose relative)

Fig. 4. Rule Schema

Interjunction rules account for all types of relative structures, including adjectives and prepositional phrases, which we group with relative clauses because we consider them all to involve a point of intersection. Nonrelatives are classified under full-junctions. Full-junction rules fully subtend the operands under one label node, whereas interjunction rules involve an interjoining effect at a point of intersection (which we call a topic). The topic may be a relative pronoun or other pro-form or may have no surface realization. Fig. 4 also shows a few examples of junction rules. The junction operations are conjunction (used for all types of coordination – “and”, “or”, “but”, etc.), adjunction (used to form all types of predicates and predications) and subjunction, which is used for all derivations and also to show set relationships in modification.

The axioms impose constraints on allowable categorial combina-

tions within the context of the rule schema. We distinguish five basic categories ( $N$ =noun,  $V$ =verb,  $A$  = adjective or adverb,  $P$  = preposition or relation, and  $E$  = empty) and eight complex categories, formed from the basic categories by adjunction. When the axioms apply to the schema, they produce about 200 theoretically valid rules; but in practice, fewer than 100 rules (which are numbered for computer implementation) are needed to account for the level **II** structures of most of natural language. This statement is based on the examination of the following languages by Junction Grammar specialists - English, Spanish Portuguese, Russian, Tahitian, German, Finnish, French, and Samoan.

It is clear that there are certain differences between the Junction Grammar model of language and other current models. One is a matter of terminology (see Fig. 5).

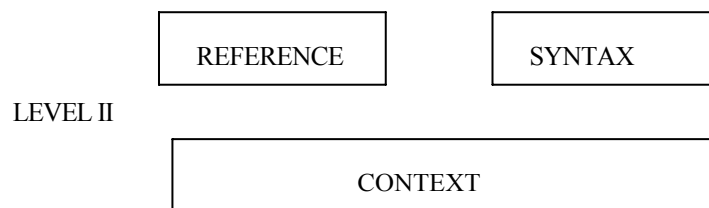


Fig. 5. *Junction Grammar Definition of Semantics*

We define semantics to be the study of level **II**, which consists of reference (semantic indices), syntax (junction rule patterns), and context (human experience acted on by logic and emotion). Many other models separate semantics and syntax, but this is because they deal with surface syntax, which we would place under the lexical and surface rule systems.

A major distinction between Junction Grammar and Transformational Grammar is that Junction Grammar uses no transformations to alter the level II representation because the junction rules, alone, seem to be sufficient to describe directly any structural pattern of natural language.<sup>3</sup> It should also be emphasized that as far as we can tell, the 200 junction rules form a universal set from which all languages draw.

<sup>3</sup> According to the informal definition of transformation suggested by J. LYONS in *Introduction to Theoretical Linguistics*, London 1968, p. 248, our lexical rules would be considered transformations. However, we do not classify Junction Grammar as a transformational grammar. See E. G. LYTLE'S thesis (listed in the Appendix I for further discussion).

Level II semantic indices are relatively language independent because in our system a sememe can be realized as a word or a phrase or even part of a word. The junction rules emphasize the generalities between languages and the lexical rules reveal the vast differences in surface manifestations.

It is also significant that the universal set of about 200 junction rules can be computed from two schema and a short list of axioms. We use a computer program to apply the axioms to the schema and produce the list of rules. Those interested in the theoretical development of the Junction Grammar model are referred to publications by E. G. LYTLE and his associates listed at the end of this paper. An important result of the closed nature of the set of junction rules is the possibility of formally defining the set of all well-formed syntacto-semantic statements. By choosing a semanticon with which to work, and then applying the lexical and surface rule systems of some language to the elements of that set, one would theoretically obtain the set of all grammatical sentences in that language (within the restrictions of the vocabulary of the semanticon). Further semantic constraints on the level II statements would then produce the set of all meaningful sentences in that language, with the same restrictions. TH. NORMAN has begun research on random sentence generation.<sup>4</sup>

## 2. *The Junction Grammar Model of Translation.*

The ability of the bilingual human to translate with ease languages that differ greatly at the surface level implies that at some level the languages do have much in common. The Junction Grammar model of language suggests that level II might capture much of that which is common between languages. The obvious experiment to test this hypothesis would be to select at random a passage in some source language, apply the surface and lexical rules to it to produce a level II WFSS, and then to run that WFSS through the lexical and surface rules of another language to produce a passage in that language.

We have found that such an approach can be useful in producing with some degree of success translations, and their quality can be sharply improved by performing some adjustments on the WFSS in level II

---

<sup>4</sup> His work is reported in our 1972 symposium report. See Appendix. I.

before lexicalization in the target languages. These adjustments we call transfers between languages, not to be confused with transformations within the grammar of a single language. Junction Grammar, it must be remembered, does not use transformations at level II within the description or generation of a single language.

We call the above approach "translation by analysis, transfer, and synthesis" (see Fig. 6).

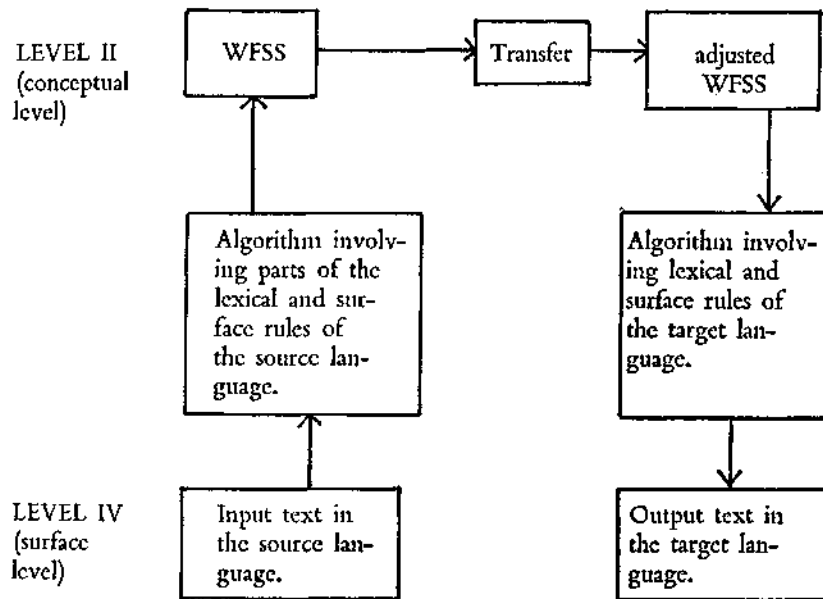


Fig. 6. *The Junction Grammar Model of Translation*

The merits of such a system are clear: First, the task of translation is modularized, facilitating the design of computer subroutines to accomplish specific tasks. Second, such a design is suited for an efficient system to translate from or to several languages because many of the routines can be language-independent and, therefore, shared.

Most approaches to machine translation use some process of neutralization. In our case, this process is the formation of an intermediate level II representation. The question is how much neutralization should occur, and in what form neutralized language should be represented. Perhaps time and experimentation will provide definitive answers to these questions. In the meantime, however, we will review the nature



of our neutralization process by discussing the differences we attempt to neutralize.

Natural languages differ in many ways. Junction Grammar suggests a categorization of these differences (see Fig. 7):

Language Differences to be Neutralized	Rule Systems Involved in Neutralization
1) Different word orders for sentences of the same meaning	L-Rules
2) Different patterns of inflection.	L-Rules
3) Different choice of words for the same concept.	L-Rules
4) Different syntactic properties of words with the same concept.	L-Rules and J-Rules
5) Source language idiomatic expressions which must be translated as a unit.	L-Rules
6) Rejection of certain junction rules as productive in a given language.	J-Rules

Fig. 7. Neutralization of Language Differences by Junction Grammar

1) different word orders for sentences of the same meaning (e.g. color adjectives are preposed in English and postposed in French);

2) different patterns of inflection (e.g. possessive adjectives depend on the gender of the possessor in English but the gender of the possessee in French);

3) different choice of words for the same concept (e.g. *tree* in English but *arbre* in French – the most obvious difference between languages);

4) different syntactic properties of words with a similar concept (e.g. in English *please* takes a direct object – *That pleases the girls* – but the French translation *plaire* takes an indirect object – *cela plait à la fille*);

5) source language idiomatic expressions which must be translated as a unit (e.g. *he kicked the bucket*);

6) rejection of certain junction rules as productive in a given language (e.g. English accepts  $N^* N(PV + N/L) = . N$  as in *He came, which surprised me*, but Japanese rejects this rule in preference to  $N^* SV = . N$  as in *It surprised me that he came*).

The first attempts at mechanical translation dealt mainly with the

most obvious difference between languages, namely difference (3), different choice of words for the same concept. But a simple word-for-word approach cannot entirely neutralize even difference (3) because of the problem of multiple meanings of a single word. Even words with one-word equivalents usually have several translations, depending on context, and many words must be translated by a phrase to reflect one or more of their meanings.

The Junction Grammar approach neutralizes difference (1) (differing word orders) automatically by the application of the lexical ordering rules to the level II representation. Difference (2) (differing patterns of inflection) is neutralized automatically by the application of the lexical agreement rules to the nodes of the level II diagram. And difference (3) (differing word choice) is neutralized by the lexical matching rules applied to the level II sememes which are free of the ambiguities found at level IV. The neutralization of differences (4), (5), and (6) (differing syntactic properties of words, idioms and rule productivity) are facilitated by the standardized form of the level II representation.

### 3. *An Application of the Junction Grammar Model to Mechanical Translation with Human Interaction.*

The Brigham Young University Language Research Center has chosen to apply the Junction Grammar model to the development of a system of mechanical translation from English simultaneously into several languages. Our tentative initial goal is to have an operational interactive system for evaluation within five years.<sup>5</sup> This system will require human interaction to complete the analysis phase. Longer range goals include the expansion of the system to at least twenty major languages of the world, preliminary work in voice-to-voice translation, and development of the logic component.

We are presently operating an experimental version of the system on the University's IBM 360/50. Most of our programming is done in PL/I although some subroutines are written in assembly language. Our present system requires about 200k bytes of main storage and one 2314 disk pack for various files.

---

<sup>6</sup> We have so far worked with synthesis in Spanish, Portuguese, German, French, and Japanese.

One of the most difficult problems encountered in implementing the Junction Grammar model in the form of working programs was how to best represent a WFSS in the machine. We finally decided on a method where the nodes and operations of each branching tree are represented as one linear table in postfix notation with cross references to the semantic indices and features, which are stored in a separate array.

A brief description of the analysis, transfer and synthesis programs will further explain our approach. In analysis, as mentioned previously, we are using an interactive approach. A sentence is read in and analyzed by general algorithms, but the machine is not always able to determine how certain words are related structurally, or which semantic index is to be assigned to a word in a given context. In order to resolve ambiguities in these cases, the present program queries the (machine) operator through a typewriter or video terminal. While the operator is responding with the appropriate answer, the program allows other programs to use the central processing unit.

Plans for future development of analysis include the implementation of a memory net simulating human experience to be used as an aid in correctly analyzing ambiguous sentences.

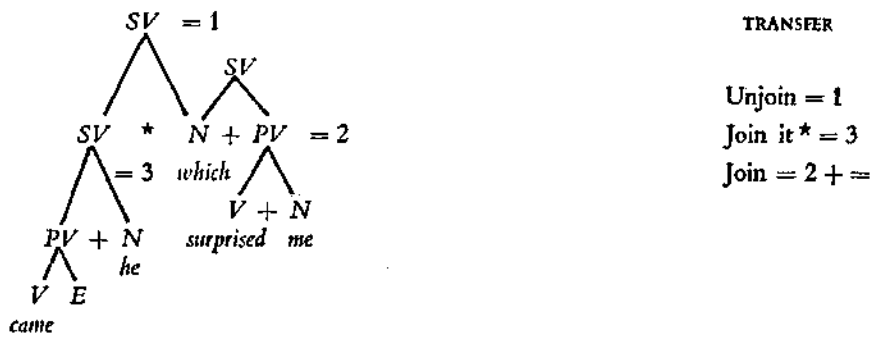
Although analysis is currently a fairly laborious process, and will be until the logic component becomes highly developed, two points should be kept in mind:

- 1) The human operator who is assisting the machine with analysis need not know the foreign language into which the text is to be translated.
- 2) Once an input sentence has been converted to a level II WFSS, it is used as input to the transfer-synthesis program for any language which is a part of the translation system.

After analysis is completed, the transfer program retrieves one WFSS at a time from a file created by analysis, and prepares it to be used as input to some synthesis program. The transfer program next takes one entry at a time from the sememe vector and examines it for needed transfers. If the entry is a non-empty semantic index, a special file is checked to see if a transfer is needed. If the entry points to the label of a junction pattern, a transfer program is called to determine whether the rule which formed that pattern is productive in the target language. In either case, when the need for a transfer is detected, the transfer is retrieved from another file in the form of a list of linguistic commands. These commands are written in our own linguistic programming language, called "transfer-language", which is embedded in PL/I.

The major advantage of using transfer-language commands on level II representations, rather than manipulating surface-level sequences with some string-manipulating language, is that level II is much more regular and well-defined than level IV. Level II capitalizes on the generalities in language and allows adjustment routines to be written in a much more general fashion than they could be in level IV. For example, all inflected forms of a concept share the same semantic index, and all segments of discourse with the same structure are grouped by their common rule number. Fig. 8 gives several examples of the use of this language.

*He came, which surprised me. → It surprised me that he came.*



Note: Words are used instead of indices for convenience only.

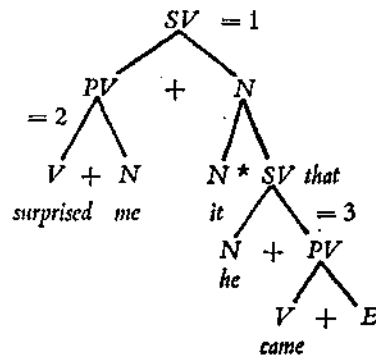
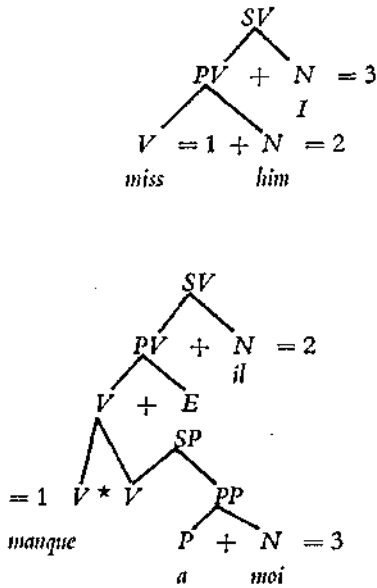


Fig. 8a. An Example of Transfer and the Use of "Transfer Language"

I miss him. → Il me manque.  
 (He misses to me.)

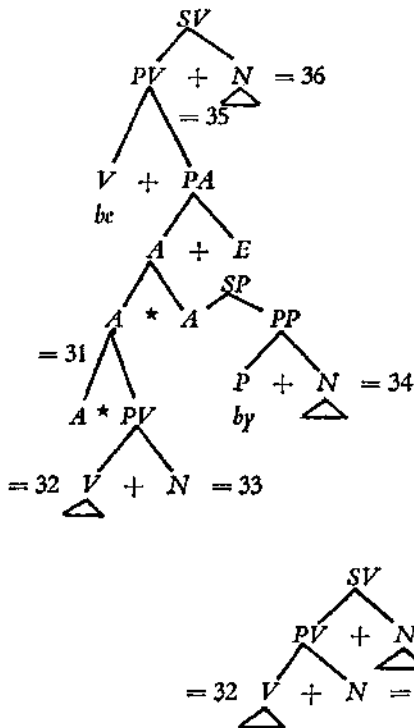


**TRANSFER**

- \*Assume = 1 is set to "miss"
- Lct = 2 be Y(label(= 1))
- Lct = 3 be Y(label(label(= 1)))
- Replace = 3 with = 2
- Join node E
- Replace = 3 with =
- Join to + = 3
- Join = 1 \$ =

Fig. 8b. An Example of Transfer and the Use of "Transfer Language"

Simple Passive → Active



**TRANSFER**

- \*Replace passive subject
- \*With logical subject
- Join node E = 40
- Replace = 36 with = 40
- Replace = 40 with = 34
- \*Build new predicate
- Replace = 32 with 0
- Join = 32 + = 36
- \*Replace old predicate with \*new
- Replace = 35 with =

Fig. 8c. An Example of Transfer and the Use of "Transfer Language"

After transfer, each WFSS, being tagged for a specific language, is passed on to the synthesis programs. The synthesis programs retrieve the appropriate WFSS's and synthesize them into the target languages. Perhaps the most significant aspect of synthesis is that the programs for the various languages have the same supervisory program and utility routines, varying only in the language-specific subroutines.

To date, we have found the Junction Grammar model to be a very adequate theoretical base for the development of machine-assisted translation programs, allowing modularity of design, and helping to avoid ad hoc solutions.

## APPENDIX I.

### *1971 Interim Report*

- 1) A. MELBY, *Toward a Formalization of the Theory.*
- 2) D. GIBB, *An Application of the Model to the Automatic Analysis of Language.*
- 3) A. MELBY, B. THOMPSON, *An Application of the Model to the Automatic Synthesis of Language.*
- 4) E. LYTLE, *Conclusions, Recommendations and Proposals.*

### *1972 Symposium*

- 1) R. L. BAIRD, *Essentials of Junction Grammar.*
- 2) A. K. MELBY, *A Formalization of Junction Grammar.*
- 3) L. S. SMITH, *The Evolution of a Computer Model of Automatic Translation Based on Junction Grammar.*
- 4) R. B. THOMPSON, *Some Basic Principles of Computerized Language Analysis.*
- 5) L. M. SCOTT, *Analysis by Resolution of Syntactic Fragments.*
- 6) D. GIBB, *Transfer-Structural Modification in Language Translation.*
- 7) M. STRONG, *Machine Translation and German Verb-Class Systems.*
- 8) R. MILLETT, *The Generation of Articles in Spanish Synthesis.*
- 9) T. NORMAN, *Random Generation of Well-Formed Syntactic Statements.*
- 10) F. BILLINGS, T. THOMSON, *Some Proposals for Ordering Well-Formed Syntactic Statements.*
- 11) B. L. BROWN, *Speech Synthesis as a Tool in Psychological Assessment.*
- 12) G. LYTLE, *Remarks on the Structure of Discourse.*
- 13) S. FLORENCE, *A Proposal for Teaching Grammar in Grammar School.*
- 14) R. OLSEN, *Junction Grammar: Its Application to Foreign Language Pedagogy.*

### *1973 Symposium*

- 1) R. B. PURVES, *Toward the Machine Recognition of Spoken Language.*
- 2) B. BROWN, *Verbal and Vocal Indices of Personality.*
- 3) R. B. THOMPSON, *Three Approaches to Translation.*
- 4) CH. BUSH, *Synthesis without a Computer.*

- 5) A. K. MELBY, *Interactive Sentence Analysis and Transfer*.
- 6) F. BILLINGS, *Procedures for Ordering Well-Formed Syntactic Statements*.
- 7) E. G. LYTLE, *An Analysis of Non-Verbal Participles*.
- 8) D. GIBB, *English Analysis: What You See Is What You Get*.
- 9) R. OLSEN, *The Effect of Language Trees on Foreign Language Acquisition*.
- 10) L. S. SMITH, *Development of a Text Concordance and Statistics System*.
- 11) S. P. GAMERO, *Transculturalization in the Translation Process*.

*Other Sources*

- F. H. BILLINGS, *An Application of the Junction Grammar Representation as a Basis for Generating German Word Order Patterns* (thesis), Brigham Young University, forthcoming.
- D. K. GIBB, *An Application to Mechanical Translation of a Variable Recursive Algorithm Based on the Operations of Union and Intersection* (thesis), Brigham Young University, 1970.
- E. G. LYTLE, *A Grammar of Subordinate Structure in English*, The Hague, 1973.
- E. G. LYTLE, *Structural Derivation in Russian* (dissertation), University of Illinois (Champaign-Urbana), 1971.



## APPENDIX II

(Concerning the formalization of Junction Grammar which was mentioned in the main paper in the discussion of the closed nature of the set of junction patterns).

### A FORMALIZATION OF JUNCTION GRAMMAR

#### INTRODUCTION

This appendix is a description of recent work on the formalization of Eldon Lytle's junction grammar. In mathematics, a formal and rigorous course in calculus is often delayed until some informal and intuitive concepts are internalized by the student. Likewise, this paper is not intended as an introduction to junction grammar but rather as a formal re-examination of certain aspects of Dr. Lytle's theory. Therefore I assume on the part of the audience a good understanding of Dr. Lytle's junction rules and the branching diagrams they produce. I also assume an understanding of basic mathematical concepts such as set, subset, mapping and tuple.

Before proceeding we will mention a few of our reasons for attempting a formalization of junction grammar. First, researchers who apply junction grammar to automatic language processing need a precise and explicit description of the set of all well-formed junction grammar structural diagrams. Automatic analysis programs must be able to build structural diagrams. Transfer programs must be able to modify them. And synthesis programs must be able to interpret them. A human can do all these things using a less formal presentation of the theory than found in this paper by using his vast experience with language. But a machine which is to do the same thing requires much more explicit and detailed instructions than does the human because the machine lacks human experience. This is the first reason for attempting a formalization of junction grammar.

A second reason for a formalization is to point out logical inconsistencies within the system by allowing formal conclusions. Sometimes the system will produce ridiculous results which can be traced back to some false assumption. Initial attempts at formalizing junction grammar have already pointed out inconsistencies in such areas as the exact nature of the dominant nodes in branching diagrams, the definition of the linking operation,<sup>1</sup> and details of the general ordering algorithm.

---

<sup>1</sup> In this paper I will use "operation" in the mathematical sense of a mapping from a set (say 'A') crossed with itself (i.e.  $A \times A$ ) to itself (i.e.  $A \times A \rightarrow A$ ).

A third reason for attempting a formalization is the hope that new understandings will be gained through formal predictions. Surely, the most significant example of this in junction grammar was the prediction of a whole range of junction rules based on the pattern  $X X S$  seen in relative clauses. Details of this significant observation are found in *A Grammar of Subordinate Structures in English* by Eldon G. Lytle. More recently, simple principles of ordering have been shown to apply to non-Indo-European languages which were not even considered in the initial formulation of the ordering principles.

In summary, the more formal and explicit a theory is, the more easily it can be applied to automatic language processing. Also, a formal theory points out both inconsistencies and generalities which may otherwise have been missed.

#### NOTATION AND TERMINOLOGY

We will now note a few differences in notation and terminology between junction grammar as taught in linguistics classes and as presented in this formalization. Structural diagrams are drawn on the blackboard in the form of branching diagrams. In this paper, structural diagrams will be sets of strings of symbols. For our structural diagrams we will use the term "Well-Formed Syntactic Statement", which is sometimes written acronymically as WFSS. Both branching diagrams and well-formed syntactic statements are made up of nodes and operations. There are two ways to represent nodes, one more formal than the other. Fig. 1 shows these two notations for the fourteen nodes of junction grammar.

Each node, when written in formal notation, is of the form  $aX$  (see Fig. 2a).

The  $X$  (called the nucleus) is  $E$ ,  $U$ ,  $N$ ,  $V$ ,  $A$ , or  $P$ ; and the  $a$  (called the coefficient of adjunction) is 1, 2, or 3. Note that the co-efficient of adjunction must be one if the nucleus is  $E$  or  $U$ . This is because units labelled  $1E$  or  $1U$  cannot be used as primary operands in adjunction rules.

That all nodes are of the form  $aX$  is useful in discussing adjunction rules. For example, the rules  $1V + 1N \Rightarrow 2V$  and  $2V + 1N \Rightarrow 3V$  and many others fall into the pattern shown in Fig. 2b.

This generalized adjunction rule indicates that adjunction raises the coefficient of adjunction, as would be expected.

As mentioned before, structural diagrams are made up of nodes and operations. Having discussed the two notations for nodes, we will now turn to the various notations for the operations. The junction operations are written the same in all three forms of structural diagrams: branching diagrams, labelled brackets, and sets of strings. However, the labelling operation is represented differently in each of the three forms of structural diagrams.

Name	Informal symbol	Formal symbol
empty node	<i>E</i>	<i>1E</i>
uncategorized node	<i>U</i>	<i>1U</i>
noun	<i>N</i>	<i>1N</i>
verb	<i>V</i>	<i>1V</i>
ad	<i>A</i>	<i>1A</i>
preposition	<i>P</i>	<i>1P</i>
noun predicate	<i>PN</i>	<i>2N</i>
verb predicate	<i>PV</i>	<i>2V</i>
ad predicate	<i>PA</i>	<i>2A</i>
preposition predicate	<i>PP</i>	<i>2P</i>
noun predication	<i>SN</i>	<i>3N</i>
verb predication	<i>SV</i>	<i>3V</i>
ad preposition	<i>SA</i>	<i>3A</i>
preposition predication	<i>SP</i>	<i>3P</i>

Fig. 1. Two notations for the fourteen nodes of junction grammar

When written in formal notation, each node can be written in the following form:

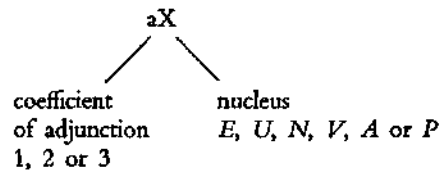


Fig. 2a.

A generalized adjunction rule

$$aX + bY = (a + 1) X$$

Examples:

$$2V + 1N = 3 V$$

$$1V + 1N = 2 V$$

Fig. 2b.

Fig. 3 shows how the labelling operation is represented in the different types of structural diagrams.

Name	Branching Diagrams	Labelled Brackets	Sets of Strings
adjunction	+	+	+
conjunction	&	&	&
subjunction	*	*	*
labelling	$\wedge$	[ ]	$\doteq$

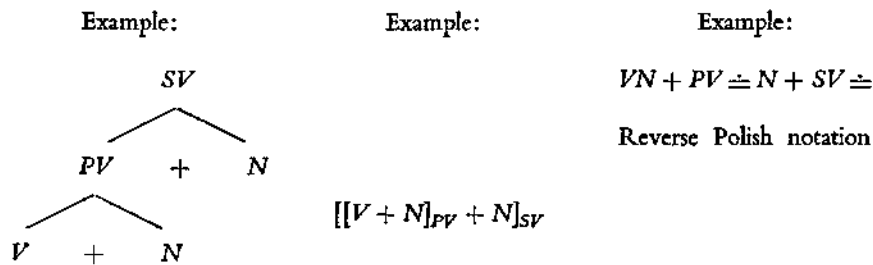


Fig. 3. Three notations for the four operations of junction grammar

Now that we have discussed the various notations for nodes and operations, we will discuss what we mean by strings. Then we will use strings to define oriented expressions and oriented expressions to define syntactic statements. Finally, we will define which syntactic statements are well-formed. This will complete the present formalization of junction grammar. Later formalizations will hopefully include theorems about well-formed syntactic statements and a description of the lexical rules which convert structural diagrams into strings of words.

### STRINGS

One definition of "string" is "a series of things arranged in or as if in a line" (*Webster's III International*). This is essentially what we mean by "string". Given some non-empty set (that is, a set with one or more elements), we will define a string to be one or more elements of that set arranged in a line. We will allow the same element to be used more than once in forming the same string. For example, given the set (Q, +, \$), Fig. 4a shows some strings which could be formed from that set.

Given the set  $\{Q, +, \$\}$  we can form, for example, the following strings:

$Q\ Q+Q$   
 $+$   
 $++$   
 $\$ Q\ Q\ \$+Q+Q\ Q\ Q$   
 $Q$   
 $\$ \$+ \$$       etc.

Fig. 4a.

We will require each string to be of finite length, but we establish no upper bound on the length of a string. Thus, a string may consist of one symbol or thousands. In writing strings, it should be made clear by commas or some other delimiter where one string ends and another begins.

It should now be clear that if we take as our set all the letters of the alphabet and such punctuation as the apostrophe and the hyphen, then the words in *Webster's III International* dictionary are simply elements of the set of strings formed from the alphabet set. Similarly, certain strings of words are sentences.

Of course, the set of all strings which can be formed from any non-empty set is infinitely large. Even if there is only one element, say  $E$ , in the set, we can still form the strings  $E, EE, EEE, EEEE$ , etc.

Given a non-empty set  $A$ , we will denote the set of all strings which can be formed from elements of  $A$  by  $A$  with an asterisk placed on the upper right corner ( $A^*$ ). This will be read "ey-star" (See Fig. 4b).

The set of all strings which can be formed from a set  $A$  will be denoted

$A^*$

Fig. 4b.

Now that we have defined strings in general, we note that we are not interested in all possible strings. We are specifically interested in strings which correspond to well-formed labelled brackets.<sup>2</sup> Therefore, we will set up the formalization so that it will distinguish between those strings which correspond to well-formed labelled brackets and those that do not.

---

<sup>2</sup> Well-formed labelled brackets normally represent the structure of phrases or clauses.

## ORIENTED EXPRESSIONS

Now we will define “oriented expression” in terms of strings. Let us begin with the set consisting of the fourteen nodes and the four operations, using the equivalence sign ( $\doteq$ ) for the labelling operation. We will call this set  $B$ , standing for basic symbols. Now let us form the set of all strings which can be formed from  $B$ , which we denote  $B^*$ . Finally, we will form the set of all subsets of  $B^*$ , which we will denote  $P(B^*)$ . Thus each element of the set  $P(B^*)$  is itself a set of strings. The empty set, denoted  $\emptyset$ , is also an element of  $P(B^*)$ , since the empty set is a subset of every set.

Now we define an oriented expression to be an ordered pair consisting of a string and a set of strings. Thus an oriented expression could be written as the ordered pair  $(\alpha \beta)$  where  $\alpha$  is a string and  $\beta$  is a set of strings, (see Fig. 5 for a few examples of oriented expressions).

An oriented expression is an ordered pair of the form

$$(\alpha \beta), \text{ where}$$

$$\alpha \in B^*, \text{ and}$$


$$\beta \in P(B^*).$$

Examples of oriented expressions:

$$(NV++ \&, \{VV^* * + \&\})$$

$$(SA SV SP P, \emptyset)$$

Example of an oriented expression which is also a wellformed syntactic statement:

$$(VN + PV \doteq N N^* N \doteq + SV \doteq,$$


$$\{AE + PA \doteq N + SA \doteq\})$$

Fig. 5.

In the last example, the arrow between the  $N$ s indicates that the two nodes are linked together. In contrast with our present practice of showing links explicitly in structural diagrams, we here suggest that links should be regarded as a device which is exterior to the structural diagram proper.

In order to explain why we have defined the oriented expressions, we must mention the notion of rank as used in junction grammar. Generally, the highest ranking unit of a structural diagram of a sentence corresponds to

the main clause of the sentence stripped of its modifiers. For example, consider the sentence *The newspaper reporter who was at the scene of the fire received third degree burns*. The highest ranking unit of this sentence's structural diagram would correspond to *The reporter received burns*.

The highest ranking unit of a well-formed syntactic statement is always one of its strings. Each well-formed syntactic statement is also an oriented expression. And each oriented expression has one string as its first element. As might be expected, we will arrange the definition of "well-formed syntactic statement" so that the first element of a well-formed syntactic statement will be its highest ranking string.

Having defined the oriented expression, we will define the syntactic statement as a subset of the oriented expression.

#### DEFINITION OF THE INITIAL SYNTACTIC STATEMENT

We will define the syntactic statement in stages. First, we will define six initial syntactic statements. The initial syntactic statements are shown in Fig. 6.

$$\begin{array}{l} (E, \emptyset) \\ (U, \emptyset) \\ (N, \emptyset) \\ (V, \emptyset) \\ (A, \emptyset) \\ (P, \emptyset) \end{array}$$

Fig. 6. *The initial syntactic statements*

When we generate a well-formed syntactic statement we always begin with syntactic statements chosen from this list of six.

We define the highest ranking string of an oriented expression to be its first element. So the highest ranking strings of the initial syntactic statements are the nodes *E*, *U*, *N*, *V*, *A*, and *P*, respectively. When one of these nodes appears as the first element of an initial syntactic statement, it is called a "terminal node". Note that only *E* and *U* are always terminal. The others (*N*, *V*, *A*, and *P*) can also appear as "label nodes" or "intersect nodes".

We will shortly add other syntactic statements to this initial list of six. One of the distinctions between syntactic statements and other oriented expressions is that syntactic statements will be made up entirely of labelled strings. A "labelled string" is simply a string for which we have defined a "label node". We define the label node of a terminal node to be the node itself. Later, when we wish to define more syntactic statements than our initial

six, we will also define what is the label node of the highest ranking string. The label node may be any of the nodes except  $E$  and  $U$ .

In summary, we have defined a set of strings ( $B^*$ ) using the nodes and operations (see Fig. 7).

nodes  $E, U, N, V, A, P, PN, PV, PA, PP, SN, SV, SA, SP$

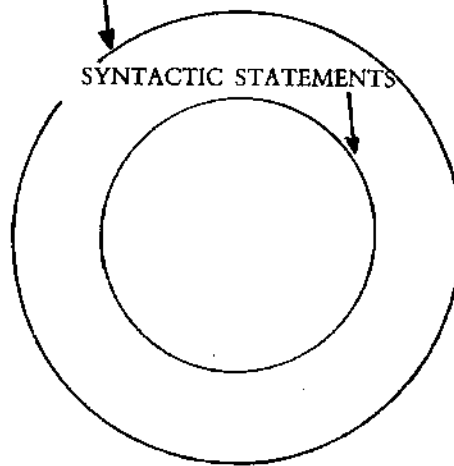
operations  $\{+, \&, *, \doteq\}$

... are used to define ...

strings  $B^*$

... which are used to define ...

ORIENTED EXPRESSIONS



$(\alpha, \beta)$

$\alpha \in B^*$

$\beta \in P(B^*)$

Fig. 7.

Then we used strings to define oriented expressions. We are now in the process of defining the syntactic statements as a subset of the oriented expressions.

As already mentioned, we call the first element of a syntactic statement the highest ranking string. We call the second element the remainder. Thus any syntactic statement is an ordered pair consisting of the highest ranking string and the remainder. We would like to add to the set of syntactic statements, but we must first define the junction rules explicitly. And in turn we must define the "junction label mapping" before we can define the junction rules.



THE JUNCTION LABEL MAPPING

The “junction label mapping” determines what will be the label node of a string formed by a junction rule. The label node of a new string is selected on the basis of the junction operation involved and the label node of the highest ranking string of the primary operand. Thus we define the “junction label mapping” in terms of two variables – a junction operation and a node. See Fig. 8 for the precise definition of this mapping, which we call  $L$ .

$$\begin{aligned}
 L (*, aX) &= aX, \text{ for any node except } 1E \text{ and } 1U. \\
 L (&, aX) &= aX, \text{ for any node except } 1E \text{ and } 1U. \\
 L (+, aX) &= (a + 1) X \text{ for any node of the form } 1 X \\
 &\text{ or } 2 X \text{ except } 1 E \text{ and } 1 U.
 \end{aligned}$$

Fig. 8. Definition of the junction label mapping  $L$

We do not define the junction label mapping on  $1E$  or  $1U$  because units with these labels are not allowed to be primary operands in junction rules. Also, we do not define  $L(+, 3X)$  because this would give rise to nodes of the form  $4X$ , which we have not found a need for.

FORMAT OF THE JUNCTION MAPPINGS

We will now define mappings which correspond to the junction rules. Each junction rule is either a full-junction rule or an interjunction rule. Fig. 9a shows the general format of these two types of junction rules.

Generalized junction rule format full-junction rules:

$$X (O Y) \dagger \doteq Z$$

Interjunction rules:

$$X * (Y / \dagger D) \dagger \doteq X$$

Fig. 9a.

We will now define the format of the corresponding junction mappings. Then we will define the mappings themselves. We define a full-junction mapping to be a 4-tuple which satisfies the following conditions:

- 1) The first element is a junction operation symbol, i.e. +, & or \*.
- 2) The second element is a node, which dictates the label node of the

primary operand. If the operation is sub-junction or conjunction, this node may be any node except  $1E$  or  $1U$ . If the operation is adjunction, the node may be any node except  $3N$ ,  $3V$ ,  $3A$ ,  $3P$ ,  $1E$ , or  $1U$ .

- 3) The third element is a node, which dictates the label node of the secondary operand(s). It may be any node.
- 4) The fourth element is a positive integer which indicates the number of iterations of junction to be performed.

Fig. 9b shows the general format of a full-junction mapping.

Junction mapping format full-junction mappings:

$$(O, aX, bY, n)$$

Interjunction mappings:

$$(f, aX, cI, bY, n)$$

Fig. 9b.

The  $O$  is the junction operation. The  $aX$  corresponds to  $X$  in the junction rule, and  $bY$  corresponds to  $Y$ . Finally,  $n$  is the iteration factor, as it is in the junction rule.

We define an interjunction mapping to be a 5-tuple which satisfies the following conditions:

- 1) The first element ( $f$ ) stands for either a  $P$  or an  $S$ , which indicates whether the adjunctive intersect (sometimes called the intersection topic) is the primary or secondary operand of the adjunction rule associated with the interjunction. This is called the format of the adjunctive intersect.
- 2) The second element is a node, which dictates the label node of the primary operand. It may be any node except  $1E$  or  $1U$ .
- 3) The third element is the node of the intersects. It may be any node except  $1E$  or  $1U$ . If the adjunctive topic is the primary operand of the adjunction, we further require that it not be  $3N$ ,  $3V$ ,  $3A$ , or  $3P$ .
- 4) The fourth element is a node, which dictates the category of the secondary operand(s). It may be any node if the format of the adjunctive intersect is the primary operand, If the intersect is the secondary operand of the adjunction, then it may not be  $1E$ ,  $1U$ ,  $3N$ ,  $3V$ ,  $3A$ , or  $3P$ .
- 5) The fifth element is a positive integer which indicates the number of iterations of interjunction which are to be performed.

See Fig. 9b, for the general format of an interjunction mapping.

$f$  is P (primary operand) or  
S (secondary operand)

$aX$  corresponds to  $X$  in the interjunction rule format.  
 $cI$  corresponds to  $Y$  in the interjunction rule format.  
 $bY$  is one adjunctive level lower than the  $D$  in the interjunction rule format, but with the same nucleus.  
 $n$  is the number of iterations.

To show how these mappings correspond to junction rules, we give a few examples in Fig. 10.

junction rule	corresponding junction mapping
$V + N \doteq PV$	$(+, V, N, 1)$
$N * N /_s SA \doteq N$	$(S, N, N, PA, 1)$
$N (\& N) \ddagger \doteq N$	$(\&, N, N, 3)$

Fig. 10.

Of course, we have not yet defined the junction mappings themselves but only their form. We will define the junction mappings themselves after we define the junction operations and the intersect nodes.

*The operations.*

For completeness, we will formally define the junction grammar operations. In isolation, all four operations act the same. Let  $O$  stand for  $+$ ,  $*$ ,  $\&$  or  $\doteq$ . Then the operation  $O$  is a mapping from ordered pairs of strings to strings. A pair of strings is simply mapped onto one string consisting of the two strings concatenated together and the operation symbol placed on the right. See Fig. 11 for an example of how junction grammar operations work.

Let  $O = +, *, \& \text{ or } \doteq$ .  
 Then  $O (S_1, S_2) = S_1 S_2 O$ ,  
 where  $S_1$  and  $S_2$  are strings.

For example,

$$\begin{aligned} & \doteq (+ (VN + PV \doteq, NN \& N \doteq), SV) \\ = & \doteq (VN + PV \doteq NN \& N \doteq +, SV) \\ = & VN + PV \doteq NN \& N \doteq + SV \doteq \end{aligned}$$

Fig. 11.

*Intersect nodes.*

An intersect node is simply a node linked to another node of the same type. That is two  $1N$  nodes or two  $3P$  nodes, etc., can be linked to form pairs of intersects. We will not specify the nature of the linking, although we will mention that in computer work, the links are simply numerical pointers. To indicate explicitly that a node is linked to another, we can write an  $1/$  after it and indicate to which node it is linked. Figure 12 shows various ways to indicate that two  $N$  nodes, for example, are intersect nodes and are linked together.

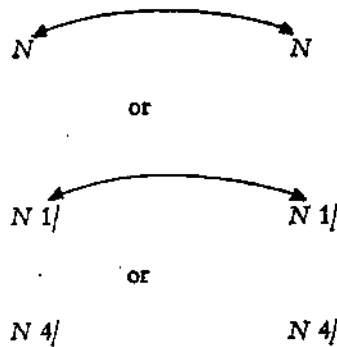


Fig. 12. *Different ways to indicate that  $N$  nodes are linked together*

The label node of an intersect node is defined to be the node itself; and thus we can add the set of all intersect nodes in the form  $(X\ 1/, \emptyset)$  to the set of syntactic statements.

*The Full-junction mappings.*

Now we are in a position to define the junction mappings explicitly. Let us begin with any particular full-junction mapping. By definition this mapping will be of the form

$$(O, aX, bY, n)$$

This mapping corresponds to the full-junction rule.

$$X (O Y) \dagger \doteq Z.$$

In describing a mapping, one generally describes the domain, then the co-domain (sometimes called the range space), and then the process of getting from the domain to the co-domain. The domain of a full-junction mapping is a set of tuples of syntactic statements. In particular, a full-junction mapping of iteration factor  $n$  is defined on a set of  $(n + 1)$  tuples of syntactic statements. Which  $(n + 1)$  tuples of syntactic statements are in the domain of a full-junction mapping depends on the values of  $aX$  and  $bY$ . An  $(n + 1)$  tuple in the domain must be compatible with  $aX$  and  $bY$ . By compatible we mean that the first element of the  $(n + 1)$  tuple must have a label node of the highest ranking string which is the same as  $aX$ , and the  $n$  other elements of the  $(n + 1)$  tuple must have label nodes of the highest ranking strings which are the same as  $bY$ . For example, consider the full-junction mapping (see Fig. 13)  $(+, 2V, 1N, 1)$ .

a junction mapping:

$$(+, 2V, 1N, 1)$$

elements of the domain are of the form

$$(\overline{2V}, \overline{1N})$$

possible elements of the domain:

$$\begin{aligned} & ((1V \ 1N + 2V \doteq, \emptyset), (1N \ 1N \ \& \ 1N \doteq, \emptyset)), \text{ or} \\ & ((1V \ 1N + 2V \doteq, \emptyset), \\ & \quad \left. (1N \ 1N \ * \ 1N \doteq, \{1A \ 1E + 2A \doteq \ 1N + 3A \doteq\}) \right\} \end{aligned}$$

which in informal notation is junction rule:  $PV + N \doteq SV$ ; input operands:

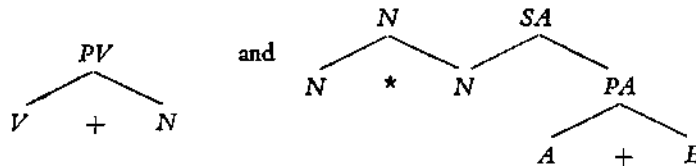


Fig. 13.

The elements of the domain of  $(+, 2V, 1N, 1)$  must be of the form  $(\overline{2V}, \overline{1N})$ , where the notation  $\overline{aX}$  stands for any syntactic statement whose highest ranking string has an  $aX$  node as its label node. For example, the two-tuple

$$((1V, 1N + 2V \doteq, \emptyset), (1N \ 1N \ \& \ 1N \doteq, \emptyset))$$

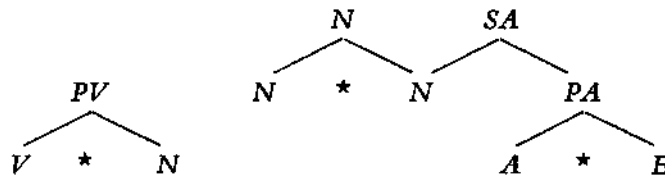
would qualify as an element of the domain of  $(+, 2V, 1N, 1)$ .  
So would the two-tuple:

$$((1V \ 1N + 2V \doteq, \emptyset), (1N \ 1N \ * \ 1N \doteq, \{1A \ 1E + 2A \doteq 1N + 3A \doteq\}))$$

Of course, this is easier to read in the informal notation junction rule:

$$PV + N \doteq SV;$$

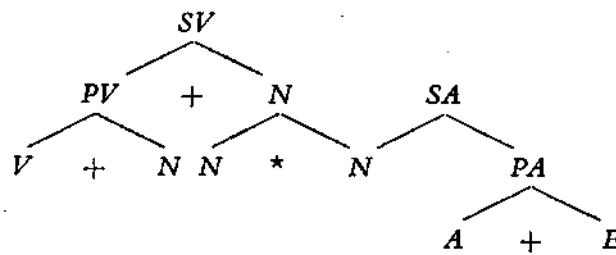
input operands:



Now we will describe the co-domain of full-junction mappings. A full-junction mapping always outputs one syntactic statement no matter how large the iteration factor  $n$  is. The output of the specific example immediately above would be the syntactic statement (see Fig. 14)

$$(1V \ 1N + 2V \doteq 1N \ 1N \ * \ 1N \doteq + 3V \doteq, \{1A \ 1E + 2A \doteq 1N + 3A \doteq\})$$

or the equivalent branching diagram



Output of the junction rule:

$$(1V \ 1N + 2V \doteq 1N \ 1N * 2N \doteq + 3V \doteq, \\ \{1A \ 1E + 2A \doteq 1N + 3A\})$$

which in branching diagram notation is:

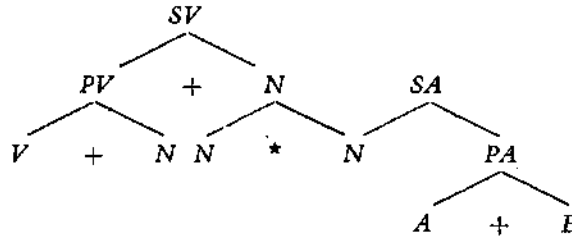


Fig. 14.

Now if we stopped with the description of the input and output, we would be no better off than if we had not begun the formalization. We must now describe precisely the process of getting to the output (see Fig. 15). Let  $S = (S_0, S_1 \dots, S_n)$  be an element of the domain of

$$(O, aX, bY, n)$$

Then each  $S_i$  ( $0 \leq i \leq n$ ) is a syntactic statement of the form  $S_i = (h_i, r_i)$ , where  $h_i$  is the highest ranking string of  $s_i$  and  $r_i$  is the set which is the remainder strings of  $s_i$ . Also,  $L(O, aX)$  is a node which is the evaluation of the junction label mapping ( $L$ ) at the junction operation ( $O$ ) and the primary operand determiner ( $aX$ ). This node is to be the label node of the highest ranking string of the output of the junction mapping. Let  $d = L(O, aX)$ . Now let  $j$  stand for our junction mapping

$$(i.e., j = (O, aX, bY, n)).$$

Then, remembering that  $S = (s_0, s_1 \dots s_n)$  is any element of the domain of  $j$  we write:

$$j(S) = (h_0 \ h_1 O \ h_2 O \ \dots \ h_n O \ d \doteq, R),$$

where  $R$  is the remainder obtained by taking the union of all the remainders of the elements of  $S$  (i.e.  $R = r_0 \cup r_1 \cup \dots r_n$ ). So we are saying formally that the junction mapping  $j$  evaluated at  $S$  consists of a highest ranking string formed by joining the highest ranking strings of the elements of  $S$  and placing a label on it, and of a remainder set consisting of the union of the remainder sets of the junction operands.

Let  $S = (S_0, S_1, \dots, S_n)$  be an element of the domain of  $(O, aX, bY, n)$ .

$S_i = (h_i, r_i), i$  from 0 to  $n$ .

$d = L(O, aX)$

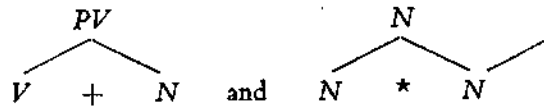
$j = (O, aX, bY, n)$

$j(S) = (h_0 h_1 \circ h_2 \circ \dots \circ h_n \circ d \doteq, R)$

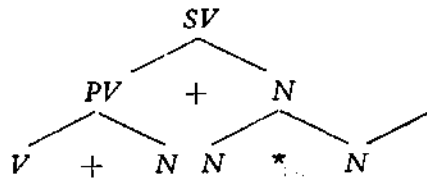
$R = r_0 \cup r_1 \cup \dots \cup r_n$

Fig. 15.

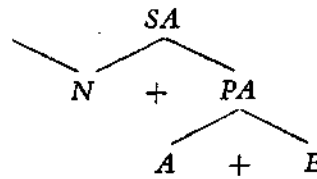
In terms of a familiar example, we join together (see Fig. 16),



to form



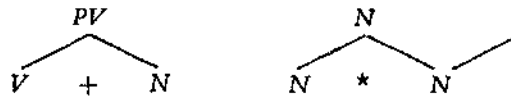
and we drag along



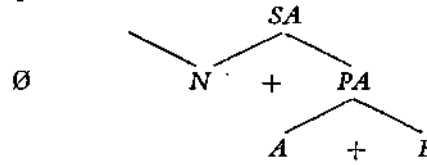
without recategorizing it.



highest ranking units of input operands



remainders of input operands



result of applying junction mapping

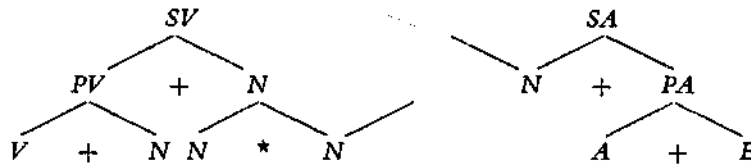


Fig. 16.

*The interjunction mappings.*

The interjunction mappings are defined in much the same way as the full-junction mappings. Let us begin with an arbitrary interjunction mapping. By definition, it can be written in the form

$$(f, aX, d, bY, n).$$

For example, see Fig. 17.

The domain of the mapping will be a subset of the  $(n + 1)$  tuples of syntactic statements restricted by the values of  $aX$  and  $bY$  in the same way as for full-junction mappings. The codomain of each interjunction mapping is the set of syntactic statements, the same as for full-junction mappings.

Now to define the process of getting the output, we let  $j$  stand for our interjunction mapping  $(f, aX, cI, bY, n)$ , and we define two full-junction mappings as follows:

$$\begin{aligned} j^* &= (*, aX, cI, n) \\ j+ &= (+, cI, bY, n), \text{ if } f=p \text{ (primary)} \\ &= (+, bY, cI, n), \text{ if } f=s \text{ (secondary)} \end{aligned}$$

Then we define the input tuples to these mappings as follows: Form  $n$  pairs of intersects of  $cI$ -nodes. For example,

$$\begin{array}{ll} cI_1 & cI_1 \\ cI_2 & cI_2 \\ cI_n & cI_n \end{array}$$

Then we define tuples of syntactic statements to be input to the mappings  $j^*$  and  $j^+$ . We do this using the contents of the input to  $j$  that is, the  $(n+1)$ -tuple  $S$ , and the pairs of intersects.

$$\begin{aligned} S^* &= (s_\emptyset, cI_1, cI_2, \dots, cI_n) \\ s_k^+ &= (cI_k, S_k), \text{ if } f = p \\ &= (S_k, cI_k), \text{ if } f = s \end{aligned}$$

for  $1 \leq k \leq n$ .

Now we perform the following junctions:

$$\begin{aligned} j^*(S^*) &= (H^*, R^*) \\ j_k^+(s_k^+) &= (h_k^+, r_k^+) \text{ (} n \text{ adjunctions)} \end{aligned}$$

Then  $j(S) =$

$$(H^*, R^* \cup h_1^+ \cup \dots \cup h_n^+ \cup r_1^+ \cup \dots \cup r_n^+)$$

an interjunction mapping

$$(f, aX, cI, bY, n)$$

which produces the following associated full-junction mappings:

$$\begin{aligned} j^* &= (*, aX, cI, n) \\ J^+ &= \begin{cases} (+, cI, bY, n), & \text{if } f = p \\ (+, bY, cI, n), & \text{if } f = s \end{cases} \end{aligned}$$

associated intersects

$$\begin{array}{cc} I_1 & I_1 \\ \vdots & \vdots \\ I_n & I_n \end{array}$$

$$S^* = (s_\emptyset, I_1, \dots, I_n)$$

$$s_k^+ = \begin{cases} (I_k, s_k), & \text{if } f = p \\ (s_k, I_k), & \text{if } f = s \end{cases} \quad k \text{ from } 1 \text{ to } n$$

$$j^*(S^*) = (H^*, R^*)$$

$$j^+(s_k^+) = (h_k^+, r_k^+)$$

$$j(S) = (H^*, R^* \cup h_1^+ \cup \dots \cup h_n^+ \cup r_1^+ \cup \dots \cup r_n^+)$$

Fig. 17.

For example, see Fig. 18.

For example, input operands

$$(N, \emptyset) \quad (A E + P A \doteq, \emptyset)$$

junction mapping

$$(S, N, N, P A, 1)$$

output

-----

$$\{N N * N \doteq, \{A E + P A \doteq N + S A \doteq\}\}$$

or

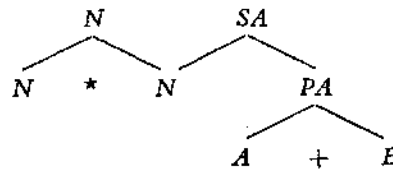


Fig. 18.

We define the label node of the highest ranking string of the output of a junction mapping to be  $L(O, aX)$  (for full-junctions and  $L(*, aX)$  for interjunction.

Therefore, we can add the output of the junction mappings to the set of syntactic statements, beginning an infinite recursive process. Thus, one syntactic statement is used to produce another which is used to form another, etc.

Next, we add further constraints on the junction mappings (See Appendix III). Then we define the well-formed syntactic statements as the output of the constrained junction mappings using only terminal statements and other well-formed syntactic statements as input. In order to obtain the full-range of well-formed syntactic statements we also define as well-formed syntactic statements those expanded syntactic statements (see Appendix III) which are produced using only well-formed syntactic statements as junction operands. Now we have defined the set of well-formed syntactic statements formally.

Further work on the formalization is now in progress, and it is hoped that it will be a significant aid to automatic language processing. This formalization of the junction rules has been used already to write a computer algorithm which randomly produces well-formed syntactic statements. It has also been used to write a program called the "WFSS building and transfer program." This program is described briefly in the main paper. The linguistic commands of the WFSS program act on WFSSs and produce WFSSs. This concept of constantly maintaining formal well-formedness throughout the execution of the program contributed significantly to the simplicity of operation of the program and to the power and simplicity of the algorithms used in its design.

APPENDIX III.

1. CONSTRAINTS ON JUNCTION MAPPING FORMAT

*Full-junction mappings.*

Write the general mapping as follows:

$$(O, aX, bY, n)$$

---

IF O is & THEN  $aX = bY$

---

IF O is \*

THEN (1) INDEPENDENT SUBJUNCTION MAPPINGS

$$b = 1; bY \neq 1E$$

(2) DEPENDENT SUBJUNCTION MAPPINGS

$$bY \neq 1E;$$

$$\text{IF } = b \neq 1 \text{ THEN } aX = bY$$


---

IF O is +

THEN (1) INDEPENDENT ADJUNCTION MAPPINGS

$$\text{IF } a = 2 \text{ THEN } bY = 1N \text{ and } aX = 2V$$

$$\text{IF } a = 1 \text{ THEN } bY = 1N, 2N, 2V, 2A \text{ or } 2P$$

(2) DEPENDENT ADJUNCTION MAPPINGS

$$\text{IF } aX = 2A \text{ or } 2P \text{ THEN } bY \neq 1E \text{ or } 1U$$

$$\text{IF } aX = 2V \text{ or } 2N \text{ THEN } bY = 1N$$

$$\text{IF } a = \text{ THEN } bY = 1N, 2N, 2V, 2A \text{ or } 2P$$


---

Interjunction mappings  $j^*$  and  $j +$  must be valid dependent full-junction mappings by above standards.

2. EXTENSION OF THE SET OF SYNTACTIC STATEMENTS

*Remark on substitution in syntactic statements.*

Let  $j$  be a junction mapping with an iteration factor of  $n$ . Let  $S$  be an  $(n + 1)$  tuple of well-formed syntactic statements which is in the domain of  $j$ . Thus  $S = (s_0, s_1, \dots, s_n)$ .

Let  $c = j(S)$ . Then suppose we substitute for  $s_i$  another well-formed syntactic statement  $s'_i$  with the same label node. This produces  $S' = (s_0, s_1, \dots, s'_i, \dots, s_n)$  and in turn  $c' = j(S')$ . Then  $c'$  will have the same label node as  $c$ . Therefore, if in any WFSS we replace what was a junction operand with any other WFSS with the same label node, the result will also be a WFSS with the same label node.

For example, in generating the deep, structural representation of the sentence

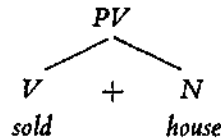
*Mr. Jones sold the house on the hill.*

we might proceed as follows:

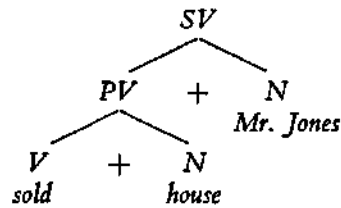
(Note that the words should really be semantic indices)

$N$	$V$	$N$
<i>Mr. Jones</i>	<i>sold</i>	<i>house</i>

Apply  $V + N \doteq PV$  to obtain



Apply  $PV + N \doteq SV$  to obtain





$$C' = (C_1, C_2'), \text{ where}$$

$$C_2' = (C_2 - (h \cup r)) \cup (h' \cup r'),$$

is defined to have the same label node as  $C$ , as one would expect, Thus we define  $C'$  to be a syntactic statement. We also allow the above process with  $m_i$  as an already expanded modifier.

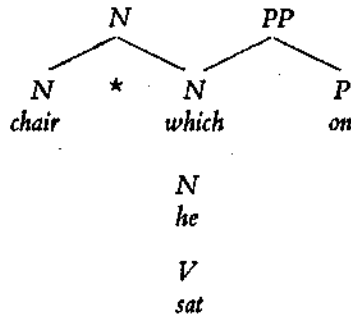
For example, in generating the diagram of the sentence

*The chair on which he sat broke*

we might proceed as follows:

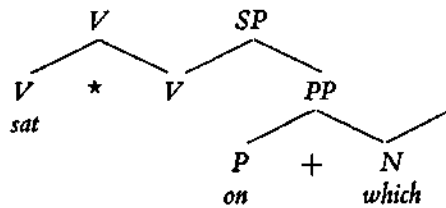
$N$        $P$   
*chair*    *on*

Apply  $N * N /_s PP \doteq N$  (i.e.  $(S, N, N, P, 1)$ ) to obtain

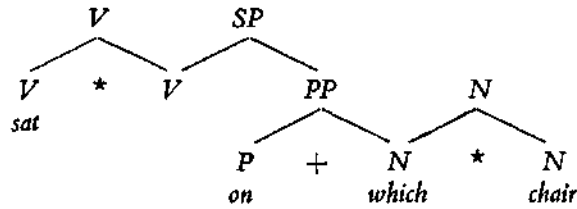


Now we would like to interjoin "sat" and "on which", but unfortunately "on which" is not the highest ranking string of "chair on which", so "on which" could not ordinarily be used as a junction operand. But "on which" is a modifier, so we expand it by applying

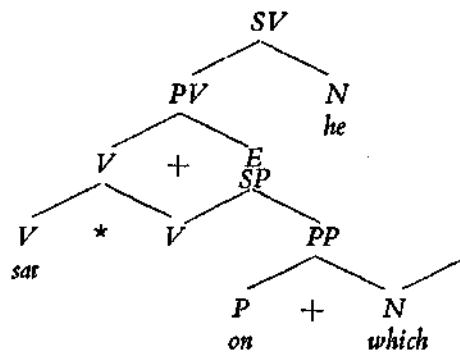
$V * V /_s SP \doteq V$ , obtaining



Now, by the expansion principle, we can put these strings back into the original WFSS, obtaining the following expanded WFSS:



Then we take the expanded modifier "sat on which" to form a triply expanded modifier by applying the rule  $V + E \equiv PV$  and  $PV + N \equiv SV$  to obtain



which can be placed back with the main unit. Finally we apply  $V + E \equiv PV$  and  $PV + N \equiv SV$  in the main clause to obtain

