I. Introduction

To a great extent, the lack of large-scale effective solutions to problems of computational linguistics can be traced to the lack of an adequate linguistic theory that could be used as a framework for computational work. Most of the linguistic theorizing that has taken place in the United States has been done under the banner of Transformational grammar. Fundamental to transformational theory is the sharp distinction between 'competence' and 'performance'.

This distinction between competence and performance provides for transformationalists the platform from which to make their statements about transformations. 'Competence', according to Chomsky is what the speaker-hearer knows about his language, as opposed to his use of that knowledge, labeled 'performance'. Chomsky includes in his discussion of what a 'performance' model should do, factors such as memory limitations, inattention, distraction, and non-linguistic knowledge. He thus leaves for 'competence' the formalization of linguistic processes representative of the speaker-hearer's knowledge of the language.

This relegation of competence makes a basic mistake however. It is necessary to differentiate the problem of formalization of linguistic knowledge and processes, i.e., competence, from the simulation of linguistic knowledge and processes, which we shall call 'simulative performance'. There is a difference between the simulation of knowledge and processes ('simulative performance') and the simulation of actual verbal behavior (Chomsky's 'performance'). It is here that we must speak, as Chomsky does, of the ideal speaker-hearer. Clearly the ideal speaker-hearer is not inattentive or distracted. He does however have memory limitations and non-linguistic knowledge. This is certainly what must be simulated as an inclusive part of linguistic theory. The kind of theory of 'performance' of which Chomsky speaks may well be in the far distant future to which Chomsky relegates it (1965). However, a theory of simulative performance is not so far off. It would seem very reasonable that the possibility of the construction of a linguistic theory that both accounts for the data and does this in such a way as to appear to be consonant with the human method for doing so, is not so remote. Clearly, such a theory must deal with non-linguistic knowledge and problems of human memory as well as the problems that Chomsky designates as 'competence'. Thus, it seems that the sharp distinction between competence and performance is artificial at best. In particular, after elimination of some of the behavioristic problems such as distraction, we can expect to find a linguistic theory that is neither one of 'competence' nor 'performance' but something in between and therefore inclusive of both.

Chomsky (1965:139) has stated:

> "Thus it seems absurd to suppose that the speaker first forms a generalized Phrase-marker by base rules and then tests it for well-formedness by applying transformational rules to see if it gives, finally, a well-formed sentence. But this absurdity is simply a corollary to the deeper absurdity of regarding the system of generative rules as a point-by-point model for the actual construction of a sentence by a speaker."

We could, on the other hand, attempt to formulate a system of rules that are a point-by-point model for the actual construction of a sentence by a speaker. Furthermore, we might expect that that system could also be a point-by-point model for the actual analysis of a sentence by a hearer. These claims, however, would be largely unverifiable except by the use of computers as simulative devices.

Chomsky (1965:141) has further stated that:

> "The grammar does not, in itself, provide any sensible procedure for finding the deep structure of a given sentence, or for producing a given sentence, just as it provides no sensible procedure for finding a paraphrase to a given sentence. It merely defines these tasks in a precise way. A performance model must certainly incorporate a grammar; it is not to be confused with grammar."

Insofar as the notion of a performance model here can be taken as being somewhere between Chomsky's notion of competence and performance, our notion of grammar also lies somewhere between Chomsky's notion of a grammar and the incorporation of a grammar.

II. Conceptual Dependency

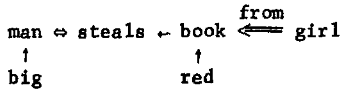
The Conceptual Dependency framework (see Schank [1969]) is a stratified linguistic system that attempts to provide a computational theory of simulative performance. The highest level of the stratificational system (similar to Lanab [1966], Sgall [1965] and others) employed by the Conceptual Dependency framework is an interlingua consisting of a network of language-free dependent concepts, where a concept may be considered to be an unambiguous word-sense, (except see Schank,[1968]). (The notion of dependency used here is related to those of Hays (1964) and Klein (1965), however, the dependencies are not at all restricted to any syntactic criterion.) The grammar of a language is defined by the framework as consisting of Realization Rules that map conceptual constructs into syntactically correct language on the 'sentential level'.

The linguistic process can be thought of, in Conceptual Dependency terms, as a mapping into and out of some mental representation. This mental representation consists of concepts related to each other by various meaning-contingent dependency links. Each concept in the interlingual network may be associated with some word that is its realizate on a sentential level.

The conceptual dependency representation is a linked network that can be said to characterize the conceptualization inherent in a piece of written language. The rule of thumb in representing concepts as dependent on other concepts is to see if the dependent concept will further explain its governor and if the dependent concept cannot make sense without its governor.

For example, in the sentence, "The big man steals the red book from the girl." the analysis is as follows: 'The' is stored for use in connecting sentences in paragraphs, i.e., 'the' specifies that 'man' may have been referred to previously. 'Big' refers to the concept 'big' which cannot stand alone conceptually. The concept 'man' can stand alone and is modified, conceptually by 'big', so it is realized in the network as a governor with its dependent. 'Steals' denotes an action that is dependent on the concept that is doing the acting. A conceptualization (a proposition about a conceptual actor) cannot be complete without a concept acting (or an attribute statement), so a two-way dependency link may be said to exist between 'man' and 'steal'. That is, they are dependent on each other and govern each other. Every conceptualization must have a two-way dependency link. 'Book' governs 'red' attributively and the whole entity is placed as objectively dependent on 'steals'. The construction 'from the girl' is realized as being dependent on the action through the conceptual object. This is a different type of dependency (denoted by ⇐). There are different forms of this 'prepositional dependency', each of which is noted by writing the preposition over the link to indicate the kind of prepositional relationship. (Although a language may use inflections or nothing at all instead of prepositions to indicate prepositional dependency, we are discussing a language-free system here and it is only the relation of the parts conceptually that is under consideration.)

The conceptual network representation of this sentence is then as follows:



The conceptual level works with a system of rules (shown in the Appendix) that operate on conceptual categories. These rules generate all the permissible dependencies in a conceptualization. Multiple combination of conceptualizations in various relationships are intended to account for the totality of human language activity at the conceptual level.

The conceptual categories are divided into governing and assisting groups:

Governing Categories

PP	An actor or object; corresponds syntactically (in English) to concrete nominal nouns or noun forms.
ACT	An action; corresponds syntactically (in English) to verbs, verbal nouns, and most abstract nouns.
LOC	A location of a conceptualization.
T	A time of conceptualization; often has variant forms consisting of parts of a conceptualization.

Assisting Categories

PA	Attribute of a PP; corresponds (in English) to adjectives and some abstract nouns.
AA	Attribute of an ACT; corresponds (in English) to adverbs and indirectly objective abstract nouns.

Thus, the categories assigned in the above network correspond closely to their syntactic correlates:

$$\begin{tikzcd}
\mathrm{PP} \arrow[r, Leftrightarrow] & \mathrm{ACT} & \mathrm{PP} \arrow[l] & \mathrm{PP} \arrow[l, Rightarrow] \\
\mathrm{PA} \arrow[u] & & \mathrm{PA} \arrow[u] &
\end{tikzcd}$$

However, in the sentence, 'Visiting relatives can be a nuisance', the syntactic categories often do not correspond with the conceptual actors and actions. The ambiguous interpretations of this sentence are:

(1) $\begin{array}{c}\text{one}\\ \Updownarrow\\ \text{visit}\\ \uparrow\\ \text{relatives}\end{array}\overset{c}{\Leftrightarrow}\text{bother}\leftarrow\text{one} \qquad\qquad \begin{array}{c}\text{PP}\\ \Updownarrow\\ \text{ACT}\\ \uparrow\\ \text{PP}\end{array}\overset{c}{\Leftrightarrow}\text{ACT}\leftarrow\text{PP}$

(Here we use the conditional present [denoted by $\underset{\Leftrightarrow}{c}$] form of the two-way dependency link, one of eight possible tense-mood forms.)

(2) $\begin{array}{l}\text{relatives}\overset{c}{\Leftrightarrow}\text{bother}\leftarrow\text{one}\\ \quad\nwarrow\!\!\searrow\\ \quad\text{visit}\end{array} \qquad\qquad \begin{array}{l}\text{PP}\overset{c}{\Leftrightarrow}\text{ACT}\leftarrow\text{PP}\\ \updownarrow\\ \text{ACT}\end{array}$

(3) $\begin{array}{c}\text{relatives}\\ \Updownarrow\\ \text{visit}\\ \uparrow\\ \text{one}\end{array}\overset{c}{\Leftrightarrow}\text{bother}\leftarrow\text{one} \qquad\qquad \begin{array}{c}\text{PP}\\ \Updownarrow\\ \text{ACT}\\ \uparrow\\ \text{PP}\end{array}\overset{c}{\Leftrightarrow}\text{ACT}\leftarrow\text{PP}$

A conceptualization is written in a conceptual dependency analysis on a straight line. Dependents written perpendicular to the line are attributes of their governor except when they are part of another conceptualization line. Whole conceptualizations can relate to other conceptualizations as actors ([1] and [3]) or attributes ([2] where ⟷ indicates that the PP at its head is the actor in a main and subordinate conceptualization [⟷ is the subordinate, written below the line]).

The Conceptual Dependency framework, at the conceptual level, is thus responsible for representing the meaning of a piece of written language in language-free terms. The representation is in terms of actor-action-object conceptualizations in a topic-comment form. Thus, words that have many syntactic forms will have only one conceptual form. This is true interlinguistically as well as intralinguistically. The meaning of a construction is always the consideration used in representation. For example, 'of' in 'a cup of water' is realized as '↔ contains X' where X is water.

$$\begin{tikzcd}
\text{cup} \arrow[d, leftrightarrow] \\
\text{contains} \\
\text{water} \arrow[u]
\end{tikzcd}$$

Similarly, in 'John's love is good', 'love' is realized conceptually as X ⇔ loves ← Y.

$$\begin{tikzcd}
\text{John} \arrow[d, Leftrightarrow, "\Leftrightarrow \text{good}"'] \\
\text{love} \\
\text{one} \arrow[u]
\end{tikzcd}$$

In order to make this framework serve as a generative theory, semantics and realization rules must be added. The realization rules are used in conjunction with a dictionary of realizates. These rules map pieces of the network in accord with the grammar. Thus, a simple rule in English might be:

$$\begin{tikzcd}
\text{PP} \\
\text{PA} \arrow[u, "= \text{Adj} + \text{N}"']
\end{tikzcd}$$

In fact, the rules are not this simple since criteria of usualness and context enter into each application of a rule. These problems are discussed elsewhere (Schank [1969]) and are not the point of this paper.

The semantics that Conceptual Dependency employs is a conceptual semantics in that it serves only to limit the range of conceptualizations in such a way as to make them consonant with experience. The form and major content of this semantics is thus universal, but since we are dealing with experience we are required to speak of someone's experience.

We will thus begin to talk about some arbitrary human's experience, or since we are dealing with a computer, we can talk of the systems' experience. Thus, the conceptual semantics consists of lists of potential dependents for any given concept. These lists are listed with respect to semantic categories if there is a generalization that can be made on that basis.

III. The Parser

The Conceptual Dependency framework is used for a natural language parser by reversing the realization rules and using the semantics as a check with reality. The system for analyzing a sentence into its conceptual representation operates on pieces of a sentence looking up the potential conceptual realizates.

All conceptualizations are checked against a list of experiences to see if that particular part of the construction has occurred before. If the construction has not occurred, or has occurred only in some peculiar context, this is noted. Thus, in the construction 'ideas ⇔ sleep', it is discovered that this connection has never been made before, and is therefore meaningless to the system. If the user says that this construction is all right, it is added to the memory; otherwise the construction is looked up in a metaphor list or aborted. The system thus employs a record of what it has heard before in order to analyze what it is presently hearing.

In order for the system to choose between two analyses of a sentence both of which are feasible with respect to the conceptual rules (see Appendix) the conceptual semantics is incorporated. The conceptual semantics limits the possible conceptual dependencies to statements consonant with the system's knowledge of the real world. The definition of each concept is composed of records organized by dependency type and by the conceptual category of the dependent. For each type of dependency, semantic categories (such as animate object, human institution, animal motion) are delimited with respect to the conceptual category of a given concept, and defining characteristics are inserted when they are known. For example, concepts in the semantic category 'physical object' all have the characteristic 'shape'. Sometimes this information is intrinsic to the particular concept involved, for example, 'balls are round'.

The semantic categories are organized into hierarchical structures in which limitations on any category are assumed to apply as well to all categories subordinate to it. The system of semantic categories and a method of constructing semantic files is discussed more fully in Schank (1969).

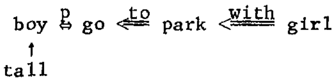
In the present system, the files are constructed by incorporating information derived from rules presented as English sentences. The program parses each of these sentences and observes which dependencies are new and then adds them to the files.

As an example of the use of the conceptual semantics, consider the parse of 'the tall boy went to the park with the girl'. At the point in the parse where the network is

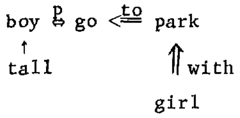
$$\begin{tikzcd}
\text{boy} \arrow[r, Leftrightarrow, "p"] & \text{go} & \text{park} \arrow[l, Rightarrow, "\text{to}"'] \\
\text{tall} \arrow[u]
\end{tikzcd}$$

we are faced with the problem of where to attach the construct $\overset{\text{with}}{\Longleftarrow}$ girl. A problem exists since at least two realization rules may apply: '$\underset{1}{\text{ACT}}$ $\underset{2}{\text{PREP}}$ $\underset{3}{\text{PP}}$: $1 \overset{2}{\Longleftarrow} 3$; '$\underset{1}{\text{PP}}$ $\underset{2}{\text{PREP}}$ $\underset{3}{\text{PP}}$: $\begin{matrix}1\\ \Uparrow 2\\ 3\end{matrix}$. The problem is resolved by the conceptual semantics. The semantics for 'go' contains a list of conceptual prepositions. Under 'with' is listed 'any movable physical object' and since a girl is a physical object the dependency is allowed. The semantics for 'park' are also checked. Under 'with' for 'park' are listed the various items that parks are known to contain, e.g., statues, junglegyms, etc. 'Girl' is not found so the network (1) is allowed while (2) is aborted.

(1)



(2)



Although '$\overset{\text{with}}{\Leftarrow}$ girl' is dependent on 'go' it is dependent through 'park'. That is, these are not isolated dependencies since we would want to be able to answer the question 'Did the girl go to the park?' affirmatively. In (2) the below-the-line notation indicates that it is the 'park with a girl' as opposed to another 'park'. Now it may well be the case that this is what was intended. The conceptual semantics functions as an experience file in that it limits conceptualization to ones consonant with the system's past experience. Since it has never encountered 'parks with girls' it will assume that this is not the meaning intended. It is possible, as it is in an ordinary conversation, for the user to correct the system if an error was made. That is, if (2) were the intended network it might become apparent to the user that the system had misunderstood and a correction could easily be made. The system would then learn the new permissible construct and would add it to its semantics. The system can always learn from the user (as described in Schank [1968]) and in fact the semantics were originally input in this way, by noticing occurrences in sample sentences.

Thus, the system purports to be analyzing a sentence in a way analogous to the human method. It handles input one word at a time as it is encountered checks potential linkings with its own knowledge of the world and past experience, and places its output into a language-free formulation that can be operated on, realized in a paraphrase, or translated.

Thus, the Conceputal Dependency parser is a conceptual analyzer rather than a syntactic parser. It is primarily concerned with explicating the underlying meaning and conceptual relationships present in a piece of discourse in any natural language. The parser described here bears some similarity to certain deep structure parsers (Kay [1967], Thorne et al [1968] and Walker [1966]) only insofar as all these parsers are concerned to some extent with the meaning of the piece of discourse being operated upon. However, the parser is not limited by the problems inherent in transformational grammar (such as the difficulty in reversing transformational rules and the notion that semantics is something that 'operates' on syntactic output). Also, the parser does not have as a goal the testing of a previously formulated grammar [as does Walker (1966) for example) so that the theory underlying the parser has been able to be changed as was warranted by obstacles that we encountered. The parser's output is a language-free network consisting of unambiguous concepts and their relations to other concepts. Pieces of discourse with identical meanings, whether in the same or different languages, parse into the same conceptual network.

The parser is being used to understand natural language statements in Colby's (1967) on-line dialogue program for psychiatric interviewing, but is not restricted to this context. In interviewing programs like Colby's, as well as in question-answering programs, a discourse-generating algorithm must be incorporated to reverse the function of the parser. The conceptual parser is based on a linguistic theory that uses the same rules for both parsing and generating, thus facilitating man-machine dialogues.

In an interviewing program, the input may contain words that the program has never encountered, or which it has encountered only in different environments. The input may deal with a conceptual structure that is outside the range of experience of the program, or even use a syntactic combination that is unknown. The program is designed to learn new words and word-senses, new semantic possibilities, and new rules of syntax both by encountering new examples during the dialogue and by receiving explicit instruction.

IV. Implementation

The parser is presently operating in a limited form. It is coded in MLISP for the PDP-10 and can be adapted to other LISP processors with minor revisions.

Rather than attaching new dependencies to a growing network during the parse, the program determines all the dependencies present in the network and then assembles the entire network at the end. Thus, the sentence 'The big boy gives apples to the pig.' is parsed into:

1) $\begin{array}{c}\text{boy}\\ \uparrow \\ \text{big}\end{array}$

2) $\text{boy} \Leftrightarrow \text{give}$

3) $\text{gives} \leftarrow \text{apples}$

4) $\text{give} \xLeftarrow{\text{to}} \text{pig}$

and then these are assembled into:

$$\begin{array}{lllllll}\text{boy} & \Leftrightarrow & \text{give} & \leftarrow & \text{apples} & \xLeftarrow{\text{to}} & \text{pig}\\ \uparrow & & & & & & \\ \text{big} & & & & & & \end{array}$$

The input sentence is processed word-by-word. After "hearing" each word, the program attempts to determine as much as it can about the sentence before "listening" for more. To this end, dependencies are discovered as each word is processed. Furthermore, the program anticipates what kinds of concepts and structures may be expected later in the sentence. If what it hears does not conform with its anticipation, it may be"confused", "surprised", or even "amused".

In case of semantic or syntactic ambiguity, the program should determine which of several possible interpretations was intended by the "speaker". It first selects one interpretation by means of miscellaneous heuristics and stacks the rests. In case later tests and further input refute or cast doubt upon the initial guess, that guess is discarded or shelved, and a different interpretation is removed from the stack to be processed. To process an interpretation, it may be necessary to back up the scan to an earlier point in the sentence and rescan several words. To avoid repetitious work during rescans, any information learned about the words of the sentence is kept in core memory.

The parse involves five steps: the dictionary lookup, the application of realization rules, the elimination of idioms, the rewriting of abstracts, and the check against the conceptual semantics.

The dictionary of words is kept mostly on the disk, but the most frequently encountered words remain in core memory to minimize processing time. Under each word are listed all its senses. "Senses" are defined pragmatically as interpretations of the word that can lead to different network structures or that denote different concepts. For example, some of the senses of "fly" are:

fly_1 - (intransitive ACT): what a passenger does in an airplane.

fly_2 - (intransitive ACT): what an airplane or bird does in the air.

fly_3 - (PP): an insect

fly_4 - (transitive ACT): what a pilot does by operating an airplane.

fly_5 - (intransitive ACT--metaphoric): to go fast.

fly_6 - (PP): a flap as on trousers.

If there are several senses from which to choose, the program sees whether it was anticipating a concept or connective from some specific category; if so it restricts its first guesses to senses in that category. Recent contextual usage of some sense also can serve to prefer one interpretation over others. To choose among several senses with otherwise equal likelihoods, the sense with lowest subscript is chosen first. Thus, by ordering senses in the dictionary according to their empirical frequency of occurrence, the system can try to improve its guessing ability.

The realization rules that apply to each word sense are referenced in the dictionary under each sense. Most of the rules fall into categories that cover large conceptual classes and are referenced by many concepts. Such categories are PP, PA, AA, PPloc, PPt, LOC, T, simply transitive ACT, intransitive ACT, ACT that can take an entire conceptualization as direct object ("state ACT") and ACT that can take an indirect object without a preposition ("transport ACT"). In contrast to most concepts, each connective (e.g., an auxiliary, preposition, or determiner) tends to have its own rules or to share its rules with a few other words.

A realization rule consists of two parts: a recognizer and a dependency chart. The recognizer determines whether the rule applies and the dependency chart shows the dependencies that exist when it does. In the recognizer are specified the ordering, categories, and inflection of the concepts and connectives that normally would appear in a sentence if the rule applied. If certain concepts or connectives are omissible in the input, the rule can specify what to assume when they are missing. Agreement of inflected words can be specified in an absolute (e.g., "plural") or a relative manner (e.g., "same tense"). Rules for a language like English have a preponderance of word order specifications while rules for a more highly inflected language would have a preponderance of inflection specifications.

Realization rules are used both to fit concepts into the network as they are encountered and to anticipate further concepts and their potential realizates in the network. When a rule is selected for the current word sense, it is compared with the rules of preceding word senses to find one that "fits". For example, if "very hot" is heard, one realization rule for "very" is:

$$\underset{0}{\text{very}} \quad \underset{1}{\text{PA}} = \begin{array}{c}1\\ \uparrow\\ 0\end{array}$$

where the tags "0" and "1" indicate the relative order of the word sense in the recognizer and identify them for reference by the dependency chart; "0" means the current word. One rule for "hot" is:

$$\underset{-1}{\text{AA}} \quad \underset{0}{\text{PA}}: \begin{array}{c}0\\ \uparrow\\ -1\end{array}$$

The program notices that "very" fits in the "-1" slot of the "hot" rule and verifies that "hot" fits in the "1" slot of the "very" rule. Therefore, the dependency suggested by the chart can be postulated for the network:

$$\begin{array}{l}\text{hot (PA)}\\ \ \uparrow\\ \text{very (AA)}\end{array}$$

After the rules for two adjacent word senses are processed, other rules are tried, and more distant word senses are checked.

Whenever a dependency is postulated, it is looked up in an idiom file to see if it is an idiom or a proper name and should be restructured. Thus, the construct:

$$\begin{array}{l}\text{make}\\ \ \uparrow\\ \text{up}\end{array}$$

is reduced to the single concept

make-up

This idiom will be detected by the parser even if several words intervene between "make" and "up" in the sentence.

After eliminating idioms from the network, there still may be constructs that do not reflect language-free conceptions. The most conspicuous cases are caused in English by abstract nouns. Most such nouns do not correspond to PP's but rather are abbreviations for conceptualizations in which the concept represented is actually an ACT or a PA.

The program treats an abstract noun as a PP temporarily in order to obtain its dependents, because abstract nouns have the syntax not of ACT's but of PP's. After obtaining its dependents, the PP is rewritten as an entire conceptualization according to rules obtained from an abstract file. These rules also specify what to do with the dependents of the PP; they may be dependent on the entire conceptualization , dependent on the ACT only, or appear elsewhere in the conceptualization.

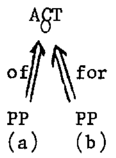
By way of example, the sentence:

Tom's love for Sue is beautiful.

leads to the following dependencies:

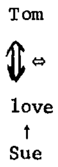
$$\begin{array}{ll} \text{love (PP)} & \text{love (PP)} \\ \Uparrow \text{ of} & \Uparrow \text{ for} \\ \text{Tom (PP)} & \text{Sue} \end{array}$$

After hearing "is", the program expects no more dependents for "love" (by a heuristic in the program), so it checks the abstract file and finds rules for "love" including:

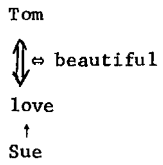


: $(a) \Leftrightarrow 0 \leftarrow (b)$

where "(a)" and "(b)" identify concepts without reference to sentential order. The network is now rewritten:



where the horizontal main link represents "is", waiting for a right-hand concept. When "beautiful" is heard, the network is complete, giving:



The network above may be realized alternatively as either of the paraphrases:

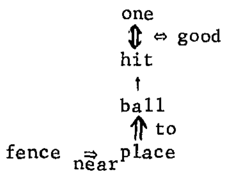
That Tom loves Sue is beautiful.

For Tom to love Sue is beautiful.

In conceptual dependency theory, connectives like "that", "for", "to", and "of" are cues to the structure of the network and need not appear in the network at all. The network above demonstrates such a situation. Conversely, portions of the network may be absent from the sentence. For example, the sentence:

It is good to hit the ball near the fence.

is parsed as:



Here, "one" and "place" are not realized. Notice that the relevant realization rule for "it" is:

$$\begin{bmatrix}\text{for} & \text{PP}\\ (a0) & (a1)\end{bmatrix} \begin{array}{cccc}\text{it} & \text{be} & \text{PA} & \text{ACT:}\\ 0 & 1 & 2 & 3\end{array} \begin{array}{l}(a1)|\text{one}\\ \Updownarrow \Leftrightarrow 2\\ 3\end{array}$$

The square brackets indicate optional words. The tags "(a0)" and "(a1)" indicate that "for" precedes the "PP" but the whole phrase may occur in any position of the construct. "(a1)|one" in the dependency chart means that if "(a1)", i.e., "for PP", is omitted, and the subject of the action is not obvious from context, then the concept "one" is to be assumed.

The conceptual network should reflect the beliefs inherent in the original discourse in a language-free representation. The interlinguistic file of conceptual semantics is checked to verify that the dependencies are indeed acceptable. This check is made after abstracts have been rewritten.

After the five parsing steps are completed, the program proceeds to the next word. At the end of the sentence, it outputs the final network in two dimensions on a printed page or on a display console.

V. Examples of Algorithm

Only a few of the relevant realization rules will be shown in the examples.

Example 1

'John saw birds flying to California.

Words	Realization Rule Patterns (for possible senses)	Dependencies (rectangle around new dependencies)
John	1: (all PP patterns)	
saw	1: (all PP patterns 2: (to see, past tense)	$\boxed{\text{John} \overset{p}{\Leftrightarrow} \text{see}} \leftarrow \text{PP}$
	$\underset{-1}{\text{PP}}\ \underset{0}{\text{ACT}}\ \underset{1}{\text{PP}}$: $-1 \Leftrightarrow 0 \leftarrow 1$	(note: "t_0" means "tense of ACT Number 0)
	$\underset{-1}{\text{PP}}\ \underset{0}{\text{ACT}}\ \underset{1}{\text{by}}\ \underset{2}{\text{PP}}$: $2 \Leftrightarrow 0 \leftarrow 1$	
	3: (to saw):... etc.	
birds	1: (all PP patterns)	$\boxed{\text{see} \leftarrow \text{birds}}$
flying	1:a) $\underset{1}{\text{PP}}\ \underset{0}{\text{ACT-ing}}$: $-1 \Leftrightarrow 0$	$\boxed{\text{birds} \Leftrightarrow \text{fly}}$ But now there are two main links on one line so go back and try as object of 'see'.
	1:b) $\underset{0}{\text{ACT}}\quad \underset{1}{\text{PP}}\quad \underset{2}{\text{ACT-ing}}$: $0 \leftarrow \begin{matrix}1\\ \Updownarrow\\ 2\end{matrix}$	
		$\text{see} \leftarrow \begin{matrix}\text{birds}\\ \Updownarrow\\ \text{fly}\end{matrix}$
	1: $\underset{-1}{\text{ACT}}\ \underset{0}{\text{to}}\ \underset{1}{\text{PP}_{\text{LOC}}}$ $-1 \xLeftarrow{0} 1$	$\text{fly} \xLeftarrow{to} \text{PP}_{\text{LOC}}$
	2: $\underset{0}{\text{to}}\ \underset{1}{\text{ACT}}$ $\Leftrightarrow 1$	
	etc.	

California	1: (all PP_{LOC} patterns)	fly $\xLeftarrow{\text{to}}$ California

Final output:

$$\begin{array}{ccccc} & & & & \text{birds}\\ \text{John} & \overset{p}{\Leftrightarrow} & \text{see} & \leftarrow & \Updownarrow \\ &&&& \text{fly} \\ &&&& \text{to}\Uparrow \\ &&&& \text{California}\end{array}$$

Example 2

'John saw Texas flying to California.'

Words	Patterns	Dependencies
John saw	1: (all PP patterns) (as above)	$\boxed{\text{John} \overset{p}{\Leftrightarrow} \text{see}} \leftarrow$ PP
Texas	1: (all PP_{LOC} patterns)	$\boxed{\text{see} \leftarrow \text{Texas}}$
flying	(as above)	Texas ⇔ fly: rejected by semantics. Laugh and go back and try (1b):

$$\begin{array}{c|c|c} \text{John} & \overset{p}{\Leftrightarrow} & \text{see}\\ & \uparrow & \\ \hline \text{John} & \overset{p}{\Leftrightarrow} & \text{fly}\end{array}$$

(rest as above)

Final Output:

$$\begin{array}{l} \text{John} \overset{p}{\Leftrightarrow} \text{see} \leftarrow \text{Texas} \\ \quad\;\; \uparrow \\ \text{John} \overset{p}{\Leftrightarrow} \text{fly} \xLeftarrow{\text{to}} \text{California} \end{array}$$

Example 3

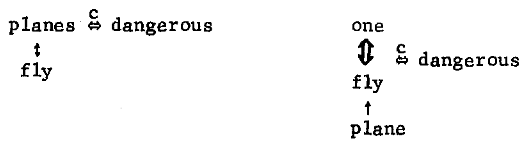
'Jane ate the hamburgers in the park.'

Words	, Patterns	Dependencies
Jane ate	1: (all PP patterns) 1: (eat - past tense)	John $\overset{p}{\Leftrightarrow}$ eat
	PP ACT PP: -1 $\overset{to}{\Leftrightarrow}$ 0 ← 1 -1 0 1	eat ← PP
	etc.	

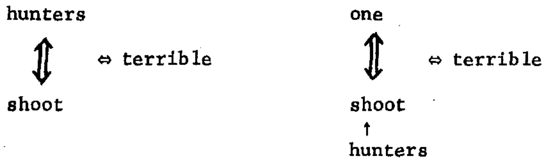
Words	Patterns	Dependencies
hamburgers	1: (all PP patterns)	eat ← hamburgers
in the park	1: ACT in PP: $-1 \overset{0}{\Leftarrow} 1$ -1 0 1	eat $\overset{\text{in}}{\Leftarrow}$ park: rejected by semantics
	2: PP in PP: $\begin{array}{c} -1 \\ 0\Uparrow \\ 1 \end{array}$ -1 0 1	$\begin{array}{l} \text{hamburgers} \\ \quad \Uparrow \text{in} \\ \quad \text{park} \end{array}$: set aside as unlikely (would accept if not other alternatives)
	3: PP ⇔ ACT in PP_{LOC}: -3 -2 -1 0 1 $\begin{array}{ccc} & -2 & \\ -3 & \Leftrightarrow & -1 \\ & \Uparrow 0 & \\ & 1 & \end{array}$	$\begin{array}{ccc} & p & \\ \text{John} & \Leftrightarrow & \text{eat} \\ & \Uparrow \text{in} & \\ & \text{park} & \end{array}$

VI. Examples of Parses

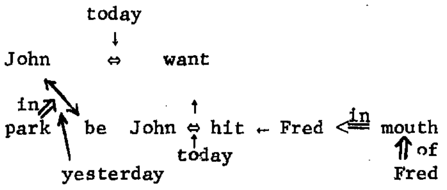
'Flying planes can be dangerous.'



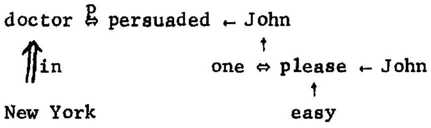
'The shooting of the hunters was terrible.'



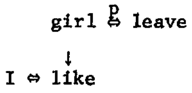
'John, who was in the park yesterday, wanted to hit Fred in the mouth today'.



'John was persuaded by the doctor in New York to be easy to please.'



'The girl I like left.'



VII. Conclusion

Before computers can understand natural language they must be able to make a decision as to precisely what has been said. The conceptual parser described here is intended to take a natural language input and place the concepts derivable from that input into a network that explicates the relations between those concepts. The conceptual network that is then formed is not intended to point out the syntactic relations present and there is some question as to why any system would want this information. Although Chomsky's deep structures convey a good deal more information than just syntactic relations, it is clear that a parser that uses deep structures for output would be oriented syntactically. We see no point in limiting our system by trying to test out a previously formulated grammar. The output of a transformational parser, while making explicit some important aspects of the meaning of the sentence, does not make explicit all the conceptual relationships that are to be found, does not limit its parses with a check with reality, and most importantly is syntax based. The parser presented here is semantics based. We aver that the system that humans employ is also semantics based. It seems clear to us that our parser satisfies the requirements that a parser must satisfy and in so doing points out the advantages of regarding language from a Conceptual Dependency point of view.

APPENDIX

I. Conceptual Rules (permissible dependencies):

```
PP ⇔ ACT; PP ⇔ PP; PP ⇔ PA; ACT ← PP; ACT ⇐ PP;

PP   PP   ACT   PA   AA   PA   T    LOC   ACT   ⇔
⇑ ;  ↑;   ↑ ;   ↑ ;  ↑ ;       ↓;   ⇓ ;   ↑ ;   ↑; ⇕⇔.
PP   PA   AA    PA   PA   PP   ⇔    ⇔     ⇔     ⇔
```

II. Realization Rules

There are about 100 of these rules, presently. A few are shown here.

```
PP ACT : 1 ⇔ 2 ;   PP ACT to ACT : 1 ⇔ 2 ;   PA PP : 2 ;
 1  2               1  2     3          ↑       1  2    ↑
                                      1 ⇔ 3            1


ACT PP PP : 1 ⇐ 2 ← 3 ;   PP Prep PP : 1
 1  2  3                  1   2    3   ⇑2 ;
                                       3
                     3
PP ACT Prep PP : 1 ⇔ 2 ⇐4  |or 1 ⇔ 2 ;   PP PP ACT ACT: 1 ⇔ 4;
1   2   3   4                   ⇑3       1  2   3   4     ↓
                                4                      2 ⇔ 3

                    1
ACT PP ACT-ing :    ↑     ;  PP who ACT ACT : 1 ⇔ 3
 1  2   3         2 ⇔ 3      1       2   3    ↕
                                              2
```

III. A Sample of the Conceptual Semantics for 'ball'.

ball,

inanimate motion object

←PA		⇚ PP		⇔ ACT	
size	any	in	phys obj	specific	bounce
shape	round	on	phys obj	motion object	roll, come, spin
color	any	for	phys obj	concrete	fall, hit ...
texture	usually smooth	by	place	any	begin, cause ...
elasticity	bounces	of	animal		
		at	no		
		to	no		

Bibliography

1. Chomsky, N., Aspects of the Theory of Syntax, MIT Press, Cambridge, 1965.

2. Colby, K., and Enea, H., "Heuristic Methods for Computer Understanding of Natural Language in Context-Restricted On-Line Dialogues," Mathematical Biosciences, 1967.

3. Hays, D., "Dependency Theory: A Formalism and Some Observations", V.40, December 1964.

4. Kay, M., "Experiments with a Powerful Parser", RAND, Santa Monica, California, 1967.

5. Klein, S., "Automatic Paraphrasing in Essay Format", Mechanical Translation, 1965.

6. Lamb, S., "The Sememic Approach to Structural Semantics", American Anthropologist 1964.

7. Schank, R., "A Conceptual Dependency Representation for a Computer-Oriented Semantics", Ph.D. Thesis University of Texas, Austin 1969 (Also available as Stanford AI Memo 83, Stanford Artificial Intelligence Project, Computer Science Department, Stanford University, Stanford, California.)

8. Schank, R., "A Notion of Linguistic Concept: A Prelude to Mechanical Translation", Stanford AI Memo 75. December 1968.

9. Sgall, P., "Generation, Production and Translation", Presented to 1965 International Conference on Computational Linguistics, New York.

10. Thorne, J., Bratley, P., and Dewar, H., "The Syntactic Analysis of English by Machine", in Machine Intelligence III, University of Edinburgh, 1968.

11. Walker, D., Chapin, P., Geis, M., and Gross, L., "Recent Developments in the MITRE Syntactic Analysis Procedure", MITRE Corp., Bedford, Mass., June 1966.