Dependency Propagation: A Unified Theory of Sentence Comprehension and Generation

Kôiti Hasida and Syun Isizaki

Machine Inference Section, Information Sciences Division,
Electrotechnical Laboratory

1-1-4, Umezono Sakura-mura Niihari-gun Ibaraki 305, JAPAN
Tel: 298-54-5423, E-mail: hasida@etl.junet

ABSTRACT

The possibility is pursued that a single mental program underlies both sentence comprehension and generation. The Horn-logic formalism is exploited here to modelize the mental representation of the linguistic knowledge, as a bundle of constraints rather than as a patchwork of procedures. A notion of **dependency** in a Horn program is defined so that eliminating dependency amounts to solving the problem (of sentence comprehension or generation) represented in terms of that program. Thus, formulated is a problem-solving paradigm called **Dependency Propagation** (**DP**): Local dependency in some parts of the program invokes execution, which may cause dependency again in some neighboring parts, which in turn invokes further execution, and so on. DP subsumes both sentence comprehension and generation, because, under DP, no heuristics are necessary about when and how to use most efficiently which piece of linguistic knowledge; The major difference between the two processes is in such alleged heuristics, whereas the declarative knowledge is largely shared. Another advantage of DP is that it captures not only short-term execution but also long-term transformation of programs. Some light is thus shed upon the evolution or acquisition of the mental grammar and lexicon.

1. Introduction

Some systematic relationship must hold between sentence comprehension and generation by humans, as suggested by the following phenomena, among others. First, most naively, the language one speaks and that one hears have similar structures. Second, there is an affinity between the process of comprehension and that of generation. For instance, we often literally guess how others' speech could continue, or detect grammatical errors and semantic inconsistencies in our own speech. Third, the two processes become equally difficult in the case of, say, deep center-embedding constructions.

In order to account for these phenomena, one might hypothesize some relationship between sentence comprehension and generation; i.e., between the two (maybe the same) grammars and between the two (maybe the same, too) programs for comprehension and generation. Here we adopt the strongest hypothesis:

(1) A single mental program underlies both sentence comprehension and generation.

Let us call this the **Common Program Hypothesis** (**CPH** for short), and the mental program mentioned therein the **Common Program** (**CP**). The challenge of the current work is to figure out how CP operates, as well as to what extent CPH can be supported.

CP must be a coherent system of instructions (and/or constraints, as it will in fact turn out) rather than a patchwork of subroutines independent of each other. As an extreme instance, CP must not consist of two modules, one for comprehension and the other for generation. More precisely, CP is defined to be the maximum domain of the mind every part of which is potentially exploited in both sentence comprehension and generation. This definition ensures the existence of CP without saying anything about its coverage, whereas CPH claims that the coverage encompasses some crucial part of both sentence comprehension and generation. Another point to be drawn from this definition is that CP contains grammar rules and lexical entries as long as they have chance to be exploited in both comprehension and production.

In the following discussion, we shall formulate CP as consisting of two components. One is a declarative representation of linguistic knowledge. This representation is modelized in terms of a logic-programming formalism, and biased in favor of neither sentence comprehension nor generation. The other component of our model of CP is an interpreter of this knowledge. It is the operation of this interpreter that is called **Dependency Propagation**. This interpreter is exempt from language-specific aspects, not to speak of comprehension-specific or generation-specific aspects.

This model of CP is based upon the observation that sentence comprehension and generation make access to the same linguistic knowledge, but in different ways. For example, consider the syntactic rule about English topicalization as in: *Mary, Tom doesn't like to see.* Put declaratively, this rule might look like (2), details being omitted.

(2) A sentence S may consist of any constituent X plus a following sentence S/X which lacks X somewhere. Here X is semantically focused on.

This rule is exploited in different ways between sentence comprehension and generation. In comprehension, perhaps this rule is fully activated only when the beginning of S/X is detected. At that time X would get focused on. In typical cases of generation, on the other hand, the rule could be activated by a focused semantic element, whereby this element is first put into a linguistic expression X.

The common linguistic knowledge such as (2) would be modeled in terms of declarative rules, constraints, or the like. The apparent difference between comprehension and generation is in the manners of access to such knowledge. In the former models of language faculty (and in application programs such as those of machine translation), this difference has been stipulated in terms of comprehension- or generation-specific heuristics about when and how to activate most efficiently which piece of linguistic knowledge. For instance, a generation-oriented heuristic rule to exploit

(2) might state:

(3) If a verbalization into a sentence is currently attempted, and the topical focus is upon a part of the input semantic content, then first translate the focused part into a language expression X, and next attempt to verbalize the remaining content as a sentence in which X is missing somewhere.

The major task in our pursuit of CPH is to substitute the heuristics of this sort with a general nonbiased paradigm of problem-solving to interpret the common declarative knowledge. That is, such a paradigm should control the timing of and data-flow in the exploitation of linguistic knowledge, just the same way as those heuristics do. For instance, Prolog interpreter does not provide such a paradigm. In fact, the existing implementations of DCG (Pereira and Warren, 1980) cannot deal with sentence comprehension and generation equally efficiently; they must be biased (by virtue of procedure attachments, etc.) in favor of one or the other task, in order to work efficiency. DP will be proposed later as a candidate for the desired paradigm.

Such a pursuit of CPH should be qualified, however, because there are some good reasons to conclude that some of those heuristics should survive for the sake of processing efficiency, and hence that CP does not encompass the entire language processing. For example, typical cases of Broca's aphasia exhibit so-called telegraphic speech (i.e., one which lacks grammatical markers such as inflections, conjugations, prepositions, etc.), the comprehension ability remaining fairly normal. Despite the apparent inconsistency, this phenomenon is compatible with CPH. A consistent interpretation is that the function of retrieving words from meaning (plus syntactic features) do not belong to CP, and therefore may be lost without reducing the ability to listen. Other evidences, including aphasic symptoms contrasted with telegraphic speech, suggest that CP should also exclude the function of retrieving meaning from words, thus totally excluding search in the lexicon.

That CP excludes lexical retrieval is predicted a priori, by taking into account the vastness of the lexicon. In the case of comprehension, a lexical entry is considered to be retrieved with its phonological form as the key. There must be some access paths which you traverse by using phonological keys to reach desired words. These paths are not likely to be exploited in generation. Similarly, the access paths through which you find words from semantic keys need not be activated in comprehension. That is, the access paths of either direction must be out of CP. To retrieve grammar rules like (2), on the other hand, is quite another story. As is demonstrated in HG (Polloard, 1984), HPSG (Pollard, 1985), etc., the inventory of grammar rules is regarded as very small (i.e., complementation, adjunction, coordination, topicalization, and few more), when the lexicon is maximized. Such a demarcation between grammar and lexicon renders trivial the search of grammar rules. Hence the (perhaps simplifying) assumption that the access paths to grammar rules are shared between comprehension and generation would not separate the resulting model very far from the reality.

To summarize, our assumption is that CP subsumes the grammar rules, the access paths to them, and the lexicon, but not the access paths to the lexicon. The rest of the paper is concerned with how the information included in CP is put to use.

2. Constrained Patterns

Sentence comprehension is a task to figure out semantic structures of given strings of words, and sentence generation is a task in which, contrariwise, strings of words are worked out of given semantic structures. Among the currently available programming paradigms, unification seems to be most promising in order to capture this bi-directionality of data-flow in CP.

Another reason for the employment of unification in describing the flow of linguistic information is that there have been developed several unification-based grammar formalisms, such as GPSG (Gazdar, Klein, Pullum, and Sag, 1985), LFG (Bresnan, 1982), HG, HPSG, FUG (Kay, 1985), and CUG (Uszkoreit, 1986). These theories provide a basis for a representation of linguistic knowledge shared between comprehension and generation. The reader may consider that the description of the grammar and the lexicon in our model exploits the techniques in the unification-based grammars mentioned above, unless stipulated otherwise.

Here we introduce a scheme for representing linguistic information. This scheme exploits the Horn-logic programming formalism (i.e., that of Prolog), so that information could flow back and forth via unification. Ordinary patterns as in Prolog in which variables are simply indeterminate, however, are problematic in that they are lacking in expressive power. To remedy this, our scheme incorporates **constraints** on variables appearing in patterns; thus such patterns are called **constrained patterns** (formerly called **conditioned patterns** in Hasida (1986)).

A constrained pattern is a pair of a pattern (of Prolog) and a constraint. A constraint is a sequence of atomic formulas (again, of Prolog), where all the predicates heading those atomic formulas (e.g., p of an atomic formula $p(X, Y)$) are defined by Horn clauses; that is, every predicate considered here must not be system-defined. For instance,

(4) $a(f(X, Y), A, B) \triangleright p(X, A), q(Y, B).$

is a constrained pattern with pattern $a(f(X, Y), A, B)$ and constraint $[p(X, A), q(Y, B)]$, provided that predicates p and q are defined in terms of Horn clauses. Note that a constrained pattern looks just like a Horn clause, except that the pattern and the constraint are separated by $\triangleright$ rather than :-.

The semantic difference between a Horn clause and a constrained pattern is that the former expresses a scheme of logical inference, while the latter denotes a set of patterns. For example, (4) represents the set exhibited in (5), in the case where predicates p and q are defined by (6).

(5) $\{a(f(1, a), 2, b), a(f(1, c), 2, d), a(f(3, a), 4, b), a(f(3, c), 4, d)\}$

(6) $p(1, 2).$ $p(3, 4).$ $q(a, b).$ $q(c, d).$

When some predicate in the constraint has a recursive definition, a constrained pattern may represent a set which cannot be denoted by a finite set of patterns possibly containing variables. For instance, the constrained pattern (7) represents the set of all the lists ending with the null list (i.e., []), where predicate *list* is defined as in (8).

(7) $X \triangleright list(X).$

(8) $list([]).$ $list([A \mid X]) :- list(X).$

A number of problems can be represented as a constrained pattern; i.e., a constrained pattern is regarded as denoting the set (or a subset) of the solutions of a problem. The problem for CP to solve, for example, is represented by a constrained pattern such as shown in (9), where the predicate *constituent* is defined as in (10).

(9) $struct(Category, X, Y) \triangleright constituent(Category, X, Y).$

(10) *constituent*(*Category*, *X*, *Y*) :-
lexicon(*Category*, *X*, *Y*).
constituent(*Mother*, *X*, *Z*) :-
phrase_structure_rule(*Mother*, *LeftDaughter*, *RightDaughter*),
constituent(*LeftDaughter*, *X*, *Y*),
constituent(*RightDaughter*, *Y*, *Z*).

In short, this is a sort of DCG. The predicates *lexicon* and *phrase_structure_rule* are also defined by Horn clauses, and, as is indicated by their names, represent the access paths to the mental lexicon and the mental grammar. Variables *X*, *Y*, and *Z* in (10) denote some portions of the terminal string, whereby the last two arguments of *constituent* constitute the differential list representing the part of the terminal string which the constituent in question exhaustively dominates, as in:

(11) *constituent*(*sentence*, [*tom*, *loves*, *mary* | *X*], *X*)

3. Dependency Propagation

This section is devoted to a formulation of a problem-soling paradigm under the representation scheme just illustrated above. In this paradigm, the logical structure of the problem to be solved determines the ordering of execution to process the information represented as constraints. In contrast, the existing Prolog interpreters carry out execution simply according to the ordering in which atomic formulas happen to appear in source programs.

3.1. Dependency versus Modularity

Let us say that there is a **dependency** between two atomic formulas sharing some variable, in the sense that the instantiation of the shared variable can be licensed only through some possibly nontrivial interaction between the two formulas. For instance, $p(X, Y)$ and $q(Y, Z)$ are dependent on each other; there does not necessarily exist a pattern γ satisfying $q(\beta, \gamma)$ for every α and β satisfying $p(\alpha, \beta)$. An atomic formula some of whose arguments is not a variable involves dependency as well; i.e., the dependency between that argument and the predicate of the atomic formula. For example, there is a dependency in $p(f(X))$, in the sense that there does not necessarily exist a pattern β such that $\alpha = f(\beta)$ for every α satisfying $p(\alpha)$.

Let us say that a constraint is **modular** when it contains no dependency at all in such a sense. In order to put it more formal, let us define the notion of **superficial modularity**. A **superficially modular** constraint is one in which all the arguments of all the atomic formula are variables and no variable occurs twice. A constraint is modular iff all the relevant constraints are superficially modular; all the relevant constraints being defined to be the constraint itself, the bodies of the Horn clauses defining the predicates in the constraint (note that the body of a Horn clause is looked upon as a constraint), the bodies of the Horn clauses defining the predicates in those bodies, and so on. A constrained pattern is said to be modular when its constraint is modular. A predicate is modular when it is defined in terms of only Horn clauses whose bodies are modular constraints. For example, (4) and (7) are modular constrained patterns, when the predicates exploited there are modular, for instance being defined by (6) and (8), respectively.

3.2. A View of Problem Solving

As demonstrated above, a constrained pattern represents a solution set of a problem. As a matter of course, however, to represent does not necessarily imply to solve. Granted that the solution should also be represented in terms of a constrained pattern, that constrained pattern would be of what might be called a 'resolved form.'

A modular constrained pattern is looked upon as 'resolved.' That is, it is an almost extensional enumeration of patterns, as typically seen in (4), thus conforming to the intuition that, in general, a resolved form should enable you to enumerate the solutions in time proportional to the sum of their complexity. Consequently, a problem represented as a constrained pattern is resolved by **modularizing** that constrained pattern; i.e., transforming it into a modular equivalent. The modularization is regarded as driven by dependency: Dependency in some parts of a constraint invokes execution which in turn gives rise to dependency in the neighboring parts of the constraint, which invokes further execution, and so on until dependency disappears. We refer to this problem-solving paradigm as **Dependency Propagation (DP)**.

For example, if a problem is represented by (12), then (13) is a resolved representation, where *p* is an arbitrary unary modular predicate and *member* and *mp* are defined as in (14).

(12) $f(E, S) \rhd member(E, S), p(E)$.

(13) $f(E, S) \rhd mp(E, S)$.

(14) *member*(*E*, [*E* | *S*]). *member*(*E*, [*A* | *S*]) :-
member(*E*, *S*).
mp(*E*, [*E* | *S*]) :- *p*(*E*). *mp*(*E*, [*A* | *S*]) :- *mp*(*E*, *S*).

A constraint is modularized by means of an algorithm similar to fold/unfold program transformation. In the above example, the dependency (i.e., the double occurrence of *E*) in the constraint-part of (12) invokes unfolding of *member*(*E*, *S*), and then folding the resulting constraint into *mp*(*E*, *S*). For further details of the algorithm, see Hasida (1986) or Hasida and Sirai (1986).

Note two aspects of DP here. First, DP is optimized in the sense that it responds only to 'pressing needs' represented as dependency, thus minimizing the waste of processing. Second, according to DP, the procedure to interpret a given representation of a problem reflects more of the logical structure of that problem rather than artifacts in the representation. Compare this with Prolog interpreters, which tend to indulge in investigation of top-down hypotheses which happened to be activated on the way of the execution ordering predetermined by the ordering in the source program. Some proposals such as *freeze* (Colmerauer, 1982) are made to overcome this defect of Prolog, but they are partial solutions unlike DP.

The current problem, represented by (9) together with (10), may generally be solved by modularizing (9) after instantiating the variables *Category*, *X* and *Y* according to the given information. Put more precise, in the case of parsing, *Category* is first left indeterminate, *X* is set to be the given sentence, such as [*tom*, *loves*, *mary*]. In the case of sentence generation, the semantic part of *Category* is first instantiated to be the given semantic content (e.g., *love*(*tom*, *mary*)) of the target sentence, *X* being left uninstantiated. *Y* is initially set to nil (i.e., []) in both cases.

3.3. Compilation as Partial Computation

One might expect that (9) could be modularized in advance, so that the amount of computation is greatly saved in individual cases of comprehension and generation; a sort of precompilation in terms of partial computation. This expectation fails, however, in almost every nontrivial case. In fact, (9) has a modular equivalent only when the language in question is regular, detailed mathematical account being omitted.

Nevertheless, one can consider instead an equivalent constrained pattern which is **semi-modular**: modular except that a variable may occur more than once in a constraint iff the instantiation of that variable is influenced at most one occurrence. A variable representing a part of the terminal string is a typical example of such a variable; it appears first as the latter component of a differential list, and second as the first component of another differential list, as Y does in (10). Such a semi-modular description of a language is available through precompilation in great many significant cases; i.e., for the class of languages including all the context-free languages, such repetition languages as $\{w^n \mid w \in \Sigma^*\}$ (n is an arbitrary natural number, and Σ an arbitrary finite set of alphabets), etc.

For instance, when the language in question is determined by (15) (i.e., the grammar given in (16)), a semi-modular equivalent $c0$ of *constituent*, is obtained simply as in (17).

(15) $lexicon(f, [a \mid X], X).$
$phrase_structure_rule(g(A, B), A, B).$

(16) $f \rightarrow a, \quad g(A, B) \rightarrow A\ B$ (A and B are variables.)

(17) $c0(f, [a \mid X], X).$
$c0(g(A, B), X, Z) :- c0(A, X, Y), c0(B, Y, Z).$

Note here that the bodies of the two Horn clauses in (17) are modular except that Y appears twice in the second.

4. A Cognitive Model

Parsing and generation by modularizing (10) or (9) in a single stroke, however, fails to fit the reality with respect to the following two aspects. First, humans process sentences from left to right, not necessarily having in mind the global view of what the sentence eventually turns out to be. Second, presumably on account of the limitation on STM capacity, humans do not pay attention to every possible solution; otherwise such phenomena as garden-path sentences and resumptive pronouns would not take any place.

4.1. Left-to-Right Processing

Sentences may be formulated as processed (i.e., comprehended or generated) little by little from left to right, so that the first aspect is incorporated in the model. To illustrate this, a sentence *Tom loves Mary* is parsed as follows.

(18)
$$\begin{array}{lcl} c_0(Category, [tom \mid X], Y) & \Rightarrow & c_1(Category, X, Y) \\ c_1(Category, [loves \mid X], Y) & \Rightarrow & c_2(Category, X, Y) \\ c_2(Category, [mary], [\,]) & \Rightarrow & c_3(Category) \\ & \vdots & \end{array}$$

In general, parsing proceeds by successively semi-modularizing the constraint $[c_i(Category, [\alpha_i \mid X], Y)]$ to $[c_{i+1}(Category, X, Y)]$, for $0 \leq i < n$, n being the sentence length and α_i the i-th terminal symbol.

Generation goes similarly:

(19)
$$\begin{array}{lcl} c_0(\alpha, [A_0 \mid X], Y) & \Rightarrow & c_1(A_0, X, Y) \\ c_1(A_0, [A_1 \mid X], Y) & \Rightarrow & c_2(A_0, A_1, X, Y) \\ c_2(A_0, A_1, [A_2 \mid X], Y) & \Rightarrow & c_3(A_0, A_1, A_2, X, Y) \\ & \vdots & \end{array}$$

That is, at first the constraint $[c_0(\alpha, [A_0 \mid X], Y)]$ is semi-modularized to yield a constraint on variables A_0, X, and Y. The variables contained in α is concealed in the definition of c_1. Afterwards, $[c_i(A_0, \cdots, A_{i-1}, [A_i \mid X], Y)]$ is semi-modularized to $[c_{i+1}(A_0, \cdots, A_i, X, Y)]$, consecutively for $i = 1, 2, \cdots$.

Under this formulation of sentence generation, the whole constraint on the entire sentence should at the very beginning be fit with the semantic structure embedded in α. This result should be rejected, since it is not always possible to have the complete meaning of the entire sentence before generation begins. A further investigation presented later will overcome this weakness.

4.2. Memory Limitation

In order to capture the limitation on the STM capacity, let us assume that there is a finite bound on the amount of working memory in which to store the Horn clauses produced dynamically during comprehension and generation. Following this assumption, Horn clauses whose **activation intensity** is weaker than others are pruned off so that the remaining Horn clauses should fall together within the limited storage.

The activation intensity of a Horn clause is mentioned here as a neurophysiological metaphor. A Horn clause activates or deactivates some other Horn clauses. The strength of this (de)activation is positively correlated with the activation intensity of that Horn clause. Horn clauses competing with each other should be inhibitory against each other. For instance, different clauses defining the same predicate should tend to deactivate one another.

This paper does not go any further into the question of what activation intensity should be like. A fuller account would require something like the connectionist approach (Waltz and Pollack, 1985). In the current computer implementation, the intensity is simulated by an integer assigned to a Horn clause.

It must be emphasized that the memory limitation is crucial in our present approach to CPH. For instance, the fluency of generation follows from the memory limitation. In (19), the constraint on A_j's instantiation can be retained for only a few j such that $0 \leq j \leq i$, due to the memory limitation. Hence the determined value of A_j should have been emitted as a part of utterance except for j very near to i. Incidentally, therefore, in practice we come up with $c_{i+1}(A_k, \cdots, A_i, X, Y)$ for some k close to i, instead of $c_{i+1}(A_0, \cdots, A_i, X, Y)$.

4.3. The Head-Driven Propagation

The currently assumed DP (i.e., the exhaustive semi-modularization in precompilation and execution) eliminates every non-vacuous dependency. This is problematic, however, for several reasons that follow. First, (semi-) modularization as currently conceived tend to consume too much memory. For example, the amount of memory occupied by the constraint of any modular equivalent of the constrained pattern (20) is $O(m \times n)$, where m and n are prime to each other, and *listm* and *listn* are defined as in (21).

(20) $X \triangleright listm(X), listn(X).$

(21) $listm([\,]).$ $\quad listm([A_1, A_2, \cdots, A_m \mid X]) :- listm(X).$
$listn([\,]).$ $\quad listn([A_1, A_2, \cdots, A_n \mid X]) :- listn(X).$

Since the memory requirement of the given representation (20) with (21) is $O(m+n)$, the coding efficiency of the representation is deemed as often reduced by modularization.

Second, the current formulation tend to waste processing time, in the sense that predicates are often generated which are never referenced. For instance, if (semi-) modularization takes place when two constrained patterns $f(X) \triangleright p(X)$ and $f(Y) \triangleright q(Y)$ are unified with each other, then a new constrained pattern $f(X) \triangleright r(X)$ is yielded together with the new predicate r. If an unsuccess-

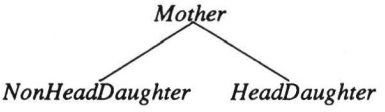
ful unification between $g(Z)$ and $f(X) \triangleright r(X)$ is attempted next, $r(X)$ is simply discarded without being exploited at all.

Third, there are many important predicates which have no semi-modular equivalents. Mathematical details being omitted, those predicates include *permutation* (a binary predicate to the effect that the two arguments are lists which are permutations of each other), *subset* (a binary predicate to the effect that the two arguments are lists and that the former represents a subset of the latter), etc.

To avoid these problems, let revise DP by relaxing the present requirement that DP semi-modularizes given constraints. To overcome the defects discussed above, DP should instead operate selectively on the parts of the given constraint which are likely to be referenced more often than others.

A typical bias of reference likelihood is found between the head and nonhead daughters in a local tree. Consider a local tree shown below, for example.

(22)



This local tree may be regarded as an instantiation of a rule such as S → NP VP and NP → Det N, in a familiar notation. When one of *Mother* and *HeadDaughter* is referenced, then it is nearly certain that the other is also referenced, because these two nodes share almost the same information. *Mother* and *HeadDaughter*, than to dependency that *NonHeadDaughter* has with *Mother* or *HeadDaughter*. Let us tentatively formulate this as the following extremely simplified form.

(23) In both sentence comprehension and generation, DP operates so that:

a. the constraint on every node is modular which dominates an exhaustively processed part of the terminal string, and

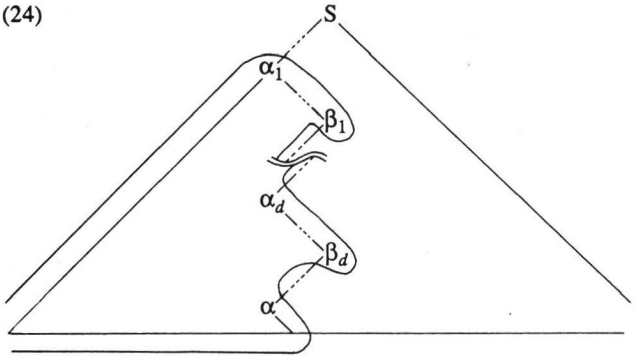
b. there is no remaining dependency between any node and its grammatical head.

Let us call this the **Head-Driven Propagation (HDP**, for short).

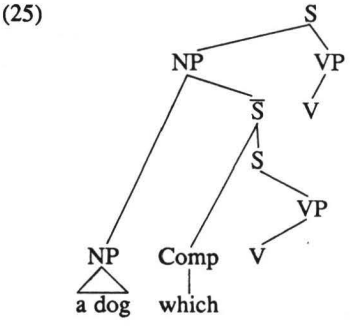
Due to (23a), HDP roughly amounts to a parallel execution of what Hasida (1985) calls the **Canonical Procedure**. The Canonical Procedure is a nondeterministic procedure to handle a coherent substructure of a sentence at a time. In general, the partial structure regarded as resolved (i.e., modularized, in our current terminology) at any stage during the execution of the Canonical Procedure looks like the one enclosed in the curve of (24) below, where the categories α_1, β_1, $\cdots$ α_d, β_d, and α are memorized in the working storage.

On account of (23b), the modular domain in the case of HDP may be a little wider than the area enclosed in the curve; That is, the former also includes the path from B_i down to its lexical head. For example, when terminal string *A dog which* is just processed, a maximal coherent structure dynamically generated would look like (25), provided that VP is a head of S and S is a head of $\overline{S}$.

(24)



(25)



Since a node and its heads greatly share information, the substantial difference between (24) and (25) is smaller than it appears.

An advantage of adopting principle-based grammars such as GB (Chomsky, 1981, 1986) and HPSG is that one can remove much of artifact in the grammar by controlling the precompilation. Such a control is possible because principle-based grammars are of the 'least precompiled' form. In HPSG, for instance, the phrase-structure rules have been abstracted away from information in the lexicon, and also from general properties of local trees, which are factored out as general principles like the Binding Inheritance Principle, the Head Feature Principle, etc. If a grammar of a more precompiled form such as GPSG were adopted, the distribution of the reference likelihood in compiled representation of grammar would be less controllable.

4.4. An Example

Now let us look at some concrete cases of DP as presently conceived in sentence processing. Since it would be too complicated as an example to precisely demonstrate how an actual natural language sentence is processed, first we examine here the language defined by (15). Several more linguistically real instances will be considered later.

Shown below is how a sentence *aaa* $\cdots$ of the language in question is parsed, following the procedure illustrated in (18).

(26)
$$c_1(g(f,B),X,Y) :- c_0(B,X,Y).$$
$$c_1(g(A,B),X,Z) :- c_1(A,X,Y),\ c_0(B,Y,Z).$$

$$c_2(g(f,B),X,Y) :- c_1(B,X,Y).$$
$$c_2(g(g(f,f),B),X,Y) :- c_0(B,X,Y).$$
$$c_2(g(A,B),X,Z) :- c_2(A,X,Y),\ c_0(B,Y,Z).$$

$$c_3(g(f,B),X,Y) :- c_2(B,X,Y).$$
$$c_3(g(g(f,f),B),X,Y) :- c_1(B,X,Y).$$
$$c_3(g(g(f,g(f,f)),B),X,Y) :- c_0(B,X,Y).$$
$$c_3(g(g(g(f,f),f),B),X,Y) :- c_0(B,X,Y).$$
$$c_3(g(A,B),X,Z) :- c_3(A,X,Y),\ c_0(B,Y,Z).$$

Horn clauses have not been pruned off here. To see what a pruning is, let us simulate the memory limitation by a simplistic requirement that the number of Horn clauses in the working memory should be no more than, say, seven, according to Miller (1958). After a possible pruning, we might be left with, for instance, the Horn clauses listed in (27):

(27) $c_3(g(f, B), X, Y) :- c_2(B, X, Y).$
$c_3(g(g(f, f), B), X, Y) :- c_1(B, X, Y).$
$c_3(g(g(f, g(f, f)), B), X, Y) :- c_0(B, X, Y).$
$c_3(g(A, B), X, Z) :- c_3(A, X, Y), c_0(B, Y, Z).$

$c_2(g(f, B), X, Y) :- c_1(B, X, Y).$

$c_1(g(f, B), X, Y) :- c_0(B, X, Y).$
$c_1(g(A, B), X, Z) :- c_1(A, X, Y), c_0(B, Y, Z).$

The Horn clauses defining c_0 are not discarded dynamically, because they have been yielded by precompilation and thus are regarded as stored in the long-term memory.

Note that every atomic formula in the bodies of the remaining Horn clauses are solvable by finite patterns, based only upon the remaining Horn clauses; e.g., $c_2(B, X, Y)$ is solvable with $B = g(f, f)$ and $X = [a \mid Y]$. This is important because a mental representation must be finite. So for instance the first Horn clause defining c_1 must be retained; otherwise no finite pattern could solve $c_1(B, X, Y)$. In order for pruning to take care of this, smaller intensity values should be assigned to Horn clauses (e.g., the second clause defining c_1) taking part in recursive definitions of predicates.

Incidentally, if a predicate is defined by only one Horn clause, it is usually more efficient (with regard to both memory and time) to unfold that predicate at all its occurrences and discard it. For instance, in (27) there remains only one clause defining c_2. Thus we obtain (28) by unfolding and eliminating c_2.

(28) $c_3(g(f, g(f, B)), X, Y) :- c_1(B, X, Y).$
$c_3(g(g(f, f), B), X, Y) :- c_1(B, X, Y).$
$c_3(g(g(f, g(f, f)), B), X, Y) :- c_0(B, X, Y).$
$c_3(g(A, B), X, Z) :- c_3(A, X, Y), c_0(B, Y, Z).$

$c_1(g(f, B), X, Y) :- c_0(B, X, Y).$
$c_1(g(A, B), X, Z) :- c_1(A, X, Y), c_0(B, Y, Z).$

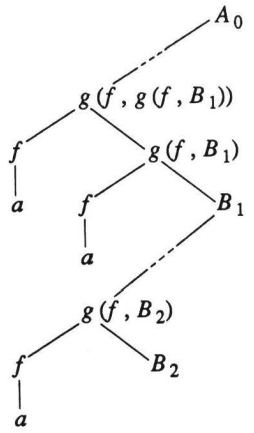
This example of DP amounts to a parallel execution of the Canonical Procedure. For instance, the four Horn clauses in (29) extracted from (28) represent together the tentative partial structure depicted in (30).

(29) $c_3(g(f, g(f, B)), X, Y) :- c_1(B, X, Y).$
$c_3(g(A, B), X, Z) :- c_3(A, X, Y), c_0(B, Y, Z).$

$c_1(g(f, B), X, Y) :- c_0(B, X, Y).$
$c_1(g(A, B), X, Z) :- c_1(A, X, Y), c_0(B, Y, Z).$

Here the node labeled $g(f, g(f, B_1))$ is on the left corner of A_0, being possibly identical to it. These two nodes are related through the second clause defining c_3 in (29), and the distance between them in a potential completion of (30) is equal to the number of times that this clause is exploited. $g(f, B_2)$ stands in the same sort of relationship with B_1 through the second clause defining c_1. (30) amounts to a snapshot just before (24) with $d = 2$. That is, α_1, β_1, α_2, and β_2 in (24) correspond to $g(f, g(f, B1))$, B_1, $g(f, B2)$, and B_2, respectively; α in (24) has no counterpart in (30). As exemplified here, the amount of memory occupied by Horn clauses representing together the partial structure in (24) is proportional to d.

(30)



4.5. Accounts of Some Linguistic Phenomena

HDP provides a measure of **transient memory load** (**TML**; i.e., the load on STM) which is finer-grained than left-branching (Yngve, 1960), center-embedding (Church, 1980), self-embedding (Miller and Chomsky, 1963), etc.

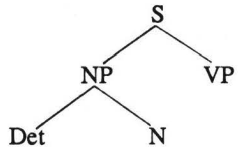
First, the complexity of a single coherent structure of a sentence is measured in terms of the memory requirement by Horn clauses needed to represent a coherent substructure of that structure at a given stage of processing. That is, TML of a maximal coherent structure is estimated to be $O(d)$ with respect to the moment depicted in (24). This measure amounts to a refinement of center-embedding; we have $d \leq \delta \leq 2d+1$, where δ is the depth of center-embedding.

Moreover, the entire memory requirement at a stage of processing in HDP captures also local structural ambiguity. Some authors have attempted to measure local ambiguity by means of the degree of lookahead (Marcus, 1980; McDonald, 1980), but the defect of such a measure is that it is not by itself sensitive to static complexity of sentences. The present framework lays a basis for talking about static complexity and local ambiguity at the same time.

HDP seems to approximate the reference likelihood of the actual mental representation of grammar. Let us consider two examples.

First, HDP is compatible with the observation that humans can 'predict' the relationship between a category and its left-corner. In English, for instance, a sentence often begins with a determiner, in constructions like (31).

(31)

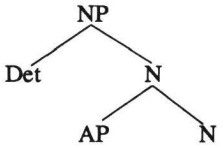


HDP is enough to 'predict' the relationship between S and Det, because, the moment Det is processed, the information that VP takes NP and NP in turn takes Det as complements (or maybe as specifiers), is exploited. It is also enough to deal with agreements of gender, number, etc. For example, the information about the agreement between the head N and Det (and AP) in (32) is incorporated in precompilation.

Second, HDP does not modularize too much. Consider resumptive pronouns, for instance. A typical context where a resumptive pronoun is found is something like (33).

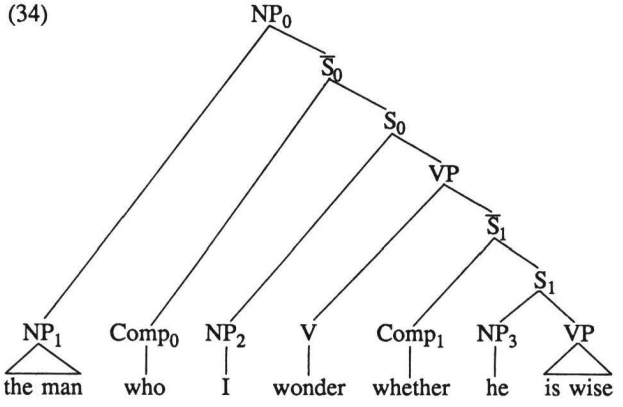
(33) the man who I wonder whether he is wise

(32)



Here, *he* is a resumptive pronoun coreferring with *the man*. HDP accounts for why such an apparently ungrammatical utterance is generated. Let us take a look at (34), the structure of (33).

(34)



HDP leaves unresolved the relationship between S_0 and NP_2 and the one between S_1 and NP_3. Therefore, it is impossible to detect the grammatical inconsistency until generation of S_1 is attempted.

McDonald (1980) proposes a different explanation of the same phenomenon, based on a deterministic model of sentence generation: a generation version of Marcus' (1980) parser. McDonald draws upon the limited lookahead presupposed in the determinism doctrine. His discussion is wrong or at best incomplete, because he fails to pay attention to how far the coming structure can be predicted via HDP rather than lookahead.

5. Final Remarks

A model has been proposed which describes both sentence comprehension and production as a single program. This program is a system of constraint rather than a sequence of instructions, and the procedure CP to interpret given information is derived from the computational structure of the task to be performed. This model exploited a problem-solving paradigm called Dependency Propagation: Dependency in the given constraint invokes execution, which proceeds so that the likelihood of reference becomes homogeneous over the whole constraint.

If we note that the likelihood of reference is related with the density of some sort of information, some part of the discussion suggests a general principle that linguistic information should be homogeneously distributed over the entire representation of the mental grammar: A principle perhaps stemming out of the more ubiquitous principle that evolution optimizes the resulting system. In the light of this, Dependency Propagation lays a promising basis upon which to talk about how cerebral coding of knowledge is reformed as new information comes in; more specifically, how an accumulation of concrete instances gives rise to abstract rules of the mental grammar.

References

Bresnan, J. (ed.) (1982) *The Mental Repesentation of Grammatical Relations,* MIT Press, Cambridge, Massachusetts.

Chomsky, N. (1981) *Lectures on Government and Binding,* Foris, Dordrecht.

Chomsky, N. (1986) *Barriers,* MIT Press, Cambridge, Massachusetts.

Church, K. (1981) *On Memory Limitations in Natural Language Processing,* MIT/LCS/TR-245, Laboratory for Computer Science, MIT.

Colmerauer, A. (1982) *Prolog II Reference Manual and Theoretical Model,* ERA CNRS 363, Groupe d'Intelligence Artificielle, Université de Marselle, Marselle.

Gazdar, G., Klein, K., Pullum, G., and Sag, I. (1985) *Generalized Phrase Structure Grammar,* Basil Blackwell, Oxford.

Hasida, K. (1985) *Bounded Parallelism: A Theory of Linguistic Performance,* Doctoral Dissertation, University of Tokyo.

Hasida, K. (1986) 'Conditioned Unification for Natural Language Processing,' *Proceedings of the 11th COLING,* pp. 85-87.

Hasida, K. and Sirai, H. (1986) 'Zyookentuki Tan-ituka (Conditioned Unification; in Japanese),' *Computer Software, Vol. 3,* pp. 28-38.

Kay, M. (1985) 'Parsing in Functional Unification Grammar,' in Dowty, D., Kartunnen, L., and Zwicky A. *Natural Language Parsing,* pp. 251-278.

Langendoen, T. (1975) 'Finite State Parsing of Phrase Structure Languages and the Status of Readjustment Rules in Grammar,' *Linguistic Inquiry, Vol. 6,* pp. 533-554.

McDonald, D. (1981) *Natural Language Production as a Process of Decision Making under Constraint,* Doctoral Dissertation, Laboratory of Computer Science, MIT.

Miller, G. (1958) 'Magical Number Seven Plus or Minus Two: Some Limits on Our Capacity for Processing Information,' *The Psychological Review, Vol. 63,* pp. 81-97.

Miller, G. and Chomsky, N. (1963) 'Finitary Models of Language Users,' in Luce, R., Bush, R., and Galanter, E. (eds.) *Handbook of Mathematical Psychology, Vol. II,* pp. 419-491, John Wiley and Sons, New York.

Pereira, F., and Warren, D. (1980) 'Definite Clause Grammar for Language Analysis – A Survey and a Comparison with Augmented Transition Networks,' *Artificial Intelligence, Vol. 13,* pp. 231-278.

Pollard, C. (1984) *Generalized Phrase-Structure Grammar, Head Gramamrs, and Natural Languages,* Doctoral Dissertation, Stanford University, Stanford, California.

Pollard, C. (1985) *Lecture Notes on Head-Driven Phrase-Structure Grammar,* Center for the Study of Language and Information, Stanford University, Stanford, California.

Uszkoreit, H. (1986) 'Categorial Unification Grammars,' *Proceedings of the 11th COLING,* pp. 187-194.

Waltz, D., and Pollack, J. (1985) 'Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation,' *Cognitive Science, Vol. 9,* pp. 51-74.

Yngve, V. (1960) 'A Model and an Hypothesis for Language Structure,' in Ferguson, D. and Slobin, D. (eds.) *Proceedings of the American Psychological Society, Vol. 104,* pp. 444-466.