Comp. & Maths. with Appls. Vol. 9, No. 1, pp. 29–43, 1983
Printed in Great Britain.

0097–4943/83/010029–15$03.00/0
Pergamon Press Ltd.

GENERATING LANGUAGE FROM CONCEPTUAL GRAPHS

JOHN F. SOWA

IBM Systems Research Institute, 205 East 42nd Street, New York, NY 10017, U.S.A.

Abstract—Conceptual graphs are a semantic representation that has a direct mapping to natural language. This article presents a universal algorithm for scanning the graphs, together with a version of augmented phrase-structure grammar for specifying the syntax of particular languages. When combined with a specific grammar, the universal algorithm determines a mapping from graphs to language with several important properties: multiple surface structures may be generated from a common underlying structure, constraints on the mapping result from the connectivity of the graphs rather than *ad hoc* assumptions, and the graphs combined with phrase-structure rules enforce context-sensitive conditions.

1. CONCEPTUAL GRAPHS

Conceptual graphs evolved as a semantic representation for natural language. The earliest forms, called *existential graphs*, were invented by the philosopher Charles Sanders Peirce[1] as a replacement for the linear notations of symbolic logic: Peirce was impressed by the diagrams of molecules in organic chemistry and believed that a graphical notation for logic could simplify the rules of inference. In linguistics, Tesnière[2] used similar notations for his *dependency grammar.* The earliest forms implemented on a computer were the *correlational nets* by Ceccato[3] and the *semantic nets* by Masterman[4]. Under various names, such as *conceptual dependency graphs*[5], *partitioned nets*[6], and *structured inheritance nets*[7], graph notations have become a popular form for representing knowledge. Although every author uses a different notation and terminology, certain themes are common to most versions:

• Concepts: Nodes of the graphs represent concepts of entities, attributes, events and states.

• Instances: Different nodes of the same concept type refer to different instances of that type, unless they are marked with a *name* or other symbol to indicate the same instance.

• Conceptual relations: Arcs of the graphs represent relationships that hold between the instances of the concepts they are linked to. Labels on the arcs indicate case relations as well as causal and logical links between propositions.

• Type hierarchy: Concepts are ordered according to levels of generality, such as COLLIE, DOG, ANIMAL, LIVING-THING, ENTITY. In various theories, the hierarchy may be a tree, a lattice, or a general acyclic graph.

Schemata: A commonly occurring pattern is represented by a stereotyped conceptual graph called a *schema.* Other terms for such patterns are *frames*, *scripts*, *MOPs* and *scenarios.*

• Inheritance: Schemata that represent properties of some type of concept are inherited by all subtypes of that concept.

Figure 1 shows a conceptual graph in the notation developed by Sowa[8]. The boxes represent concepts of entities (monkey, walnut, spoon, shell) and a concept of an action (an instance of eating). The circles represent conceptual relations: a monkey is the agent of eating, the object eaten is a walnut, the instrument of eating is a spoon, and the material of the spoon is a shell, which is a part of the same walnut that is being eaten. Although the graph notation is readable, it is difficult to type at a computer terminal; for ease of typing, the equivalent linear notation uses square brackets for concepts like [MONKEY] or [EAT] and rounded parentheses for conceptual relations like (AGNT) or (OBJ). Following is a linear representation of Fig. 1:

```
[EAT]-
  (AGNT)→[MONKEY]
  (OBJ)→[WALNUT:*x]
  (INST)→[SPOON]→(MATR)→[SHELL]←(PART)←[WALNUT:*x].
```

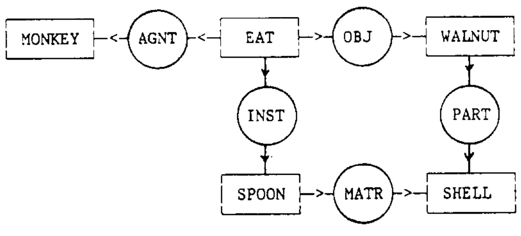

Fig. 1. A conceptual graph.

The hyphen "-" after the concept [EAT] indicates that the relations connected to [EAT] are continued on subsequent lines. If the graph has the form of a chain, it can be drawn on a single line; if it is a tree, it may be drawn on multiple lines with indentation and punctuation to show the tree structure; but if it contains a cycle, some concept node on the cycle must be repeated, and a variable symbol must be used to show cross references. In this case, the concept [WALNUT] was repeated, and the variable *x shows that both occurrences of [WALNUT] refer to the same entity.

The character strings written inside the boxes and circles of a conceptual graph are called *type labels.* Although some notations draw explicit arcs to show types and subtypes, such notations tend to look cluttered. Since the types are so pervasive, they are shown by labels inside the nodes instead of explicit arcs connecting box and circle nodes to definition nodes. A concept that has nothing but a type label (or a variable like *x) inside the box is called a *generic concept*; it refers to an unspecified individual of that type. An *individual concept* contains a serial number like #3829 or a name like Jocko after the type label; the name or serial number identifies a particular individual called the *referent* of the concept. Other kinds of referents that can occur inside concept boxes include sets, generic sets, partially specified sets and contexts containing one or more nested conceptual graphs. Following are examples of concepts with such referents and their readings in English phrases:

Kind of referent	*Notation*	*English reading*
Generic	[MONKEY]	a monkey
Individual	[MONKEY: #3829]	the monkey
Generic set	[MONKEY: {*}]	monkeys
Named individual	[MONKEY: Jocko]	Jocko
Set of individuals	[MONKEY: {Jocko, Mimi}]	Jocko and Mimi
Partially specified set	[MONKEY: {Jocko,*}]	Jocko and others

These English readings are simplified examples of the way a concept node can be mapped into an English word or phrase; the actual choice of articles, pronouns, names and modifiers depends on the entire context, including the speaker's implicit knowledge about the listener's expectations. The above table did not show a nested context as referent, since it would not fit in the available space. As an example, the next graph represents the sentence, *Sam thinks that Jocko loves Mimi*:

[PERSON: Sam]←(AGNT)←[THINK]→(OBJ)→[PROPOSITION:
    [MONKEY: Jocko]←(AGNT)←[LOVE]→(OBJ)→[MONKEY : Mimi]].

Nested contexts are used for embedded clauses, as in this example. They have also been used for logical operations, both in Peirce's existential graphs and in Hendrix's partitioned nets. Logic and model-theoretic denotations are discussed in [8]; this article concentrates on syntactic rules for mapping the graphs into natural language.

2. THE UTTERENCE PATH

The sequence of nodes and arcs that are traversed in mapping a graph to a sentence is called the *utterance path.* If conceptual graphs were always linear chains, the path could start at either end of the chain, visit each node in sequence, and utter the word that corresponds to each concept node. Since conceptual graphs are normally more complex than chains, the path would either have to skip some of the branches, or it would have to take a more circuitous walk that visits some nodes more than once. In a program for generating English, Quillian[9] chose the simple option of tracing a linear path through the graph and ignoring all side branches. McNeill[10], however, developed a psychologically motivated theory of the utterance path that permits more complex traversals. His primary motivation was to develop a theory of performance that could account not only for grammatical utterances, but also for false starts and errors.

The notion of an utterance path for scanning conceptual graphs has a great deal of explanatory power: it unifies observations about language typology and imposes strong, but well-motivated constraints on possible transformations. For co·nplex graphs, the utterance path may visit a given concept node more than once. Various languages of the world are characterized by their preference for uttering the word at the first visit to a node, the last visit, or some intermediate visit:

• *Preorder languages* utter a word at the first visit to the corresponding concept node. Biblical Hebrew, which puts the verb first and puts nouns before the adjectives, is a preorder language.

• *Postorder languages* utter a word at the last visit to the concept node. An example is Japanese, which puts the verb last, puts nouns after the adjectives, and puts postpositions after the nouns.

• *Endorder languages* utter a word at an intermediate visit to the concept node. English and French, which put verbs in the middle, are approximately endorder languages. English, however, has a postorder tendency to put nouns after the adjectives, and French has a preorder tendency to put nouns in front of the adjectives, French is a closer approximation to an endorder language, since it puts some adjectives in front of the nouns and some after them, as in *un joli chapeau rouge* instead of the English form *a pretty red hat.*

The terms *preorder, postorder,* and *endorder* are the common names of different options for scanning trees and graphs. Since preorder languages put the verb first, subject next, and object last, they are also called *VSO languages.* Postorder languages are *SOV languages,* and endorder languages are *SVO languages.* Surveys of languages around the world have found that the three patterns, VSO, SOV and SVO are common, the pattern VOS is rare, and the patterns OSV and OVS do not occur as the default patterns in any known languages[11, 12]. For emphasis, however, most languages permit optional inversions, such as the following English sentence in OSV form:

His new-found friend he took with him to the park.

Such forms, which break the normal pattern of the language, are *marked forms* as opposed to the normal, unemphatic, *unmarked forms.*

A graph with multiple branches, such as Fig. 2, illustrates the possible options for mapping a conceptual graph into a sentence. The first step is to determine a cyclic walk that starts at the main predicate [DRINK] and visits every node at least once. A sequence of concept nodes visited in such a walk would be [DRINK], [BABY], [BLITHE], [BABY], [BELLY], [FAT], [BELLY], [BABY], [DRINK], [MILK], [FRESH], [MILK], [BOTTLE], [NEW], [BOTTLE], [MILK], [DRINK]. The concepts at the ends of the branches, [BLITHE], [FAT], [FRESH] and [NEW], are only visited once; for those concepts, the corresponding word must be uttered at the moment that the walk visits the node. The other concepts are visited several times, and the words could be uttered at any visit. The following four sentences show the order of uttering the words in an endorder language such as English (1), an endorder language such as French (2), a predorder language such as Hebrew (3), and a postorder language such as Japanese (4):

(1) Blithe babies with fat bellies drink fresh milk in new bottles.
(2) Babies blithe with bellies fat drink milk fresh in bottles new.
(3) Drink babies blithe with bellies fat milk fresh in bottles new.
(4) Fat bellies with blithe babies new bottles in fresh milk drink.

The *transformations* of transformational grammar[13] result from different options in scanning the graphs. In English, the passive transformation results from following the arc to (OBJ) before the arc to (AGNT). To show that a nonstandard scan is being made, English grammar rules insert function words and inflections: *are* before the main verb, *by* before the agent, and the past participle *drunk* instead of *drink*. When the passive rules are applied in scanning Fig. 2, the following sentence is generated:

Fresh milk in new bottles is drunk by blithe babies with fat bellies.

Any of the eight relations in Fig. 2 could have been chosen as the start of an utterance path. If the relation (PART) between [BABY:{*}] and [BELLY: {*}] had been chosen as the main link, the resulting sentence would be,

Blithe babies that drink fresh milk in new bottles have fat bellies.

In English sentences generated from conceptual graphs, the verbs *be* and *have* usually correspond to relations rather than concept nodes. Those verbs occur when the main predicate is not an action, but an attribute like *new* or a noun like *belly*. In such cases, a language like Russian does not require a verb and permits forms like *Bottles new* or *At blithe babies fat bellies.* English also uses the verb *do* as a place holder: if the concept [DRINK] is expressed as the subject rather than the verb, some verb form is needed by the grammar rules; since the concept [DRINK] has already been uttered, the rules insert the verb *do* at the point where the utterance path returns to the concept [DRINK]:

Drinking fresh milk in new bottles is done by blithe babies with fat bellies.

As these examples illustrate, the same graph can be expressed in many different sentences, depending on the starting point and direction of the utterance path. Yet not all word orders are possible: the utterance path visits each node a limited number of times, and a concept can be uttered as a word only at one of those visits.

These observations can be embodied in a universal algorithm for generating sentences from conceptual graphs: start at the conceptual relation linking the subject to the main predicate, traverse every arc of every conceptual relation, and utter the word corresponding to a concept at one of the visits to its node. Each language must specify further conditions for determining which visit to a concept is the one when its word is uttered. The following six rules for translating a conceptual graph into a sentence are adapted from Sowa[14]:

(1) The utterance path must visit each concept and relation node at least once. Associated

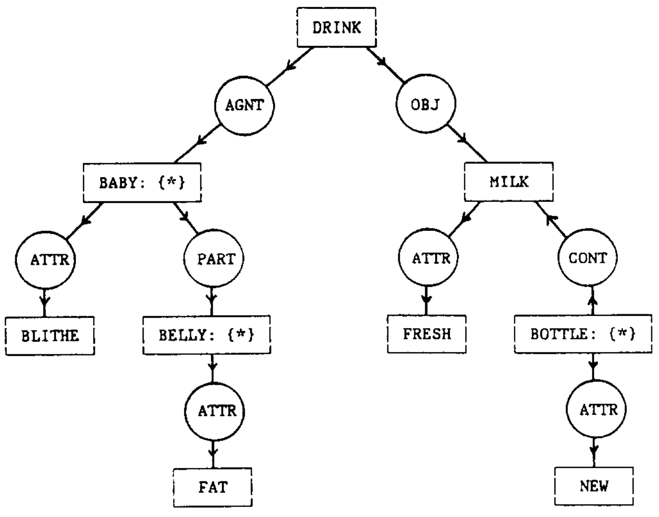

Fig. 2. A conceptual graph.

with each concept is an *utterance mark* that indicates whether the concept was uttered as a word, and with each conceptual relation a *traversal mark* that indicates whether the utterance path has traversed the arcs of that relation.

• The conceptual relation that links the subject of the sentence to the main predicate is the starting point of the path. For simple active sentences, it is a relation of type AGNT; for passive sentences, it is a relation of type OBJ or RCPT; but in general, it may be a relation of any type.

• From a concept node *c*, the path does not return to the concept from which *c* was reached until all relations linked to *c* have been traversed at least once.

• For relations that have not yet been traversed, the syntactic and stylistic rules determine which arc to follow (e.g. in English, the path must visit adverbs of manner before adverbs of time).

(2) Since the path may visit a concept several times, syntactic rules determine when the concept is uttered as a word of the sentence. Some concept types may be expressed in some languages as two words (e.g. verb-particle combinations such as *take off* or *carry out*), which may be uttered at either the same or separate visits to the concept node.

(3) Adjacent concepts in the graph should be expressed consecutively when possible. When Rule 1 requires the utterance path to take a branch or when Rule 2 skips a concept, the utterance must clarify the order of connections with markers such as intonation, inflection, function words, or conventionalized word order.

(4) For graphs nested inside the type or referent field of a concept, the utterance path enters the nested graph when the syntactic rules determine that the containing node is to be expressed. Normally, the entire nested graph is expressed at a single visit to the node, but some syntax rules (such as the *raising rules* of transformational grammer) may permit the utterance path to exit from the nested graph, visit one or more nodes in the outer graph, and then return to finish expressing the nested graph.

(5) If the graph has cycles, a concept that is reachable by two or more different paths will only be uttered once with all of its qualifiers. If syntactic rules would also express the concept at a visit reached by a different path, they must instead generate an *anaphoric expression*—either a pronoun or a short noun phrase that has the minimum number of qualifiers needed for a unique reference in the current context.

(6) The utterance path is a cyclic walk that visits every node of the graph and returns to the concept that represents the main predicate. It a graph is complicated, rules of inference may break it into multiple simpler graphs before expressing it in a sentence.

These six rules allow considerable variation of word order, but they do not permit arbitrary movement of sentence constituents. If conceptual graphs are assumed to be universal deep structures and if all languages obey the same general rules for mapping graphs to sentences, then all languages must show certain regularities in their surface structures. Following are some of the implications:

• A concept may only be expressed when the utterance path visits its node. Therefore, transformations can move a given constituent only to a limited number of positions in a sentence. Unlike transformational grammar, which requires special assumptions to rule out the unnatural transformations, the rules for scanning conceptual graphs have the correct restrictions built into them.

• Rules 1 and 6 limit the number of times a node may be visited. In particular, when the path follows a branch of modifiers, all concepts on that branch must be expressed before the walk returns to the main concept.

• Rule 3 prevents deeply embedded clauses like "The mailman that the dog that the girl that the boy loves owns bit is in the park." Such a sentence skips every other concept in a chain as the walk passes in one direction; on the way back, it skips the ones that were expressed in the first direction. Since the single function word *that* is not sufficient to show which concepts were skipped, that sentence violates the rule. Preferred versions utter adjacent concepts in adjacent content words—either in the active form "The boy loves the girl that owns the dog that bit the mailman that is in the park" or the passive form "In the park is the mailman that was bitten by the dog that is owned by the girl that is loved by the boy."

• Rule 4 for nested graphs has the same effect as the *cyclic convention* for applying transformations: all the transformations that apply to embedded clauses are performed before those that apply to higher clauses. But it slso permits *raising*[15], where constituents of a nested graph are expressed at different visits to the containing node. Consider the following graph:

[BELIEVE]→(OBJ)→[PROPOSITION: [PERSON: Ivan]←(CONT)←[KITCHEN]].

Since this graph does not include any concept that could be used as a subject for the verb *believe*, the default in English is to use an empty word like *it*:

It is believed that Ivan is in the kitchen.

But another rule of English grammar allows Ivan to be raised to the subject position of the main clause:

Ivan is believed to be in the kitchen.

In generating this sentence, the utterance path enters the nested graph to utter *Ivan*, exists from it to utter the concept [BELIEVE], and then returns to the nested graph to finish the sentence.

3. AUGMENTED PHRASE STUCTURE GRAMMAR

Since the universal rules for translating graphs to sentences allow many options, they must be supplemented with particular rules for any specific language. The specific grammar determines which arc to select first when more than one arc attached to the current node remains to be traversed. It must also specify additional markers such as word inflections and function words. The most general and flexible notation for stating the particular rules is *augmented phrase structure grammar* (APSG), which Heidorn[16, 17] developed for his NLP system. APSG rules have an underlying skeleton of phrase-structure grammar, but they are *augmented* with conditions to be tested and actions to be performed. Heidorn's NLP supports two types of APSG rules: *encoding rules* generate linear text from graphs, and *decoding rules* generate a graph representation of the meaning of a text when parsing.

A common way to write an APSG rule is to start with a standard context-free grammar rule. Then conditions for applying the rule are added on the left of the arrow, and actions are added for each nonterminal symbol on the right of the arrow. As an example, consider the rule that defines a sentence S as a noun phrase NP followed by a verb phrase VP:

S→NP VP.

This rule does not show how the NP and VP are derived from a conceptual graph, nor does it show how the person and number from the NP can affect the form of the VP. In an APSG rule, each occurrence of a nonterminal symbol like NP or VP represents a record of *attributes* that may be set of tested. The symbol on the left of the arrow represents the current goal, such as generate a sentence. That symbol is followed by *conditions* that must be true before the rule is invoked. On the right of the arrow, each symbol is followed by a list of *actions* for setting attributes or advancing the current concept node to the next node of the utterance path.Following is the general form of an APSG rule, but with English comments instead of formal symbols:

S (conditions for applying this rule)→
NP (actions for moving the current concept node to the subject and getting the person and number from the current concept)
VP (actions for moving the current concept node to the main ACT, for copying person and number from the NP record, and for copying mode and tense from the S record).

Heidorn's encoding rules have this general form, but with a more succinct, formal notation inside the parentheses. The remainder of this section develops a notation that is based on Heidoin's APSG, but with symbols and terminology adapted to conceptual graphs. The technique of associating attributes with the nonterminal symbols is closely related to *attribute grammars* [18].

APSG rules are a kind of *production rules* that are common in A.I. systems: the l.h.s. of each rule states the conditions for invoking it, and the r.h.s. states some actions to be taken, which typically cause other rules to be invoked later. The conditions and actions operate on *records* that are associated with each of the nonterminal symbols in the rules. The symbol on the left of the rule (S in the above example) is the goal to be satisfied; the symbols on the right (NP and VP in the example) are subgoals to be achieved. There is a special symbol, called the *start symbol*, whose record is created by some high-level process. That symbol (S in this example) triggers the first production rule. That rule then creates new records for NP and VP, which then trigger other rules. The process continues until terminal symbols are reached that generate the actual words of the sentence. In general, the conceptual graph contains semantic information for generating sentences, and the attributes in the records contain syntactic information as well as certain semantic information like mode and tense attributes that are passed from one record to another.

Conditions in APSG rules may test several things: attributes in the current record, the current concept or relation node of the conceptual graph, concepts and relations linked to the current node, or the utterance marks on concepts and traversal marks on conceptual relations. The conditions have the following form:

attribute "is" ["not"] test;

A rule may have zero or more of these conditions. If there are no conditions, the parentheses after the nonterminal symbol are omitted; if there are two or more conditions, they are separated by semicolons. Following are some examples of typical conditions:

type (○) is AGNT;
referent (□) is not proper name;
number is PLURAL;
tense is not present;
□→ATTR is not traversed;

In the conditions and actions, □ refers to the current concept node of the conceptual graph, and ○ refers to the current conceptual relation. The functions *type* and *referent* may be applied to one of the nodes: *type*(□) or *type*(○) retrieves the type label from the current node, and *referent*(□) retrieves the referent of the current concept. If the attribute is just a single word like *number* or *tense*, it refers to the record associated with the current nonterminal symbol; the third condition above tests whether the current record has a plural number, and the fourth one tests whether the tense attribute is missing. The symbol □→ATTR refers to the subgraph consisting of the current concept node □ linked by an arc pointing to a conceptual relation of type ATTR.

On the right side of the arrow, three kinds of actions may occur: assignment, move and mark. The assignment assigns values to the attributes in a record, move specifies the direction of movement along the utterance path, and mark indicates which nodes had already been visited.

• When an APSG rule is invoked, none of the records for the symbols on the right of the arrow have any associated attributes. An assignment action causes a new attribute to be created for a record and assigns it a value:

voice := ACTIVE;
number := number of *referent*(□);
tense := tense of VP;

The first assignment simply causes the voice attribute of the current record to have the value ACTIVE. The second one copies the number of the referent of the current concept to the number attribute of the current record. The third one copies the tense of a VP record to the tense of the current record; such copies may only be made from a record of a nonterminal symbol in the current rule that occurs *before* the symbol to which the copy is being made.

• Move causes the symbol □ or ○ to advance to the next node of the utterance path. All records in a rule start with the node □ at the same point; a move action for a given record affects only that record and not any other record in the rule.

```
move AGNT→□;

move OBJ←□;
```

The first move causes the current concept node to become the one linked to the arc pointing *away* from the relation of type AGNT. The second one causes the current concept to be the one linked to the arc pointing *towards* the relation of type AGNT.

• Mark sets the utterance mark on a concept or the traversal mark on a conceptual relation:

```
mark □→ATTR traversed;

mark □ uttered;
```

The first one sets the traversal mark on the relation of type ATTR that is linked to the current concept node, and the second sets the utterance mark on the current concept node.

The symbol □ is the equivalent of a *cursor* that advances from left to right when a linear language is being parsed. Since conceptual graphs are not linear, □ is not automatically advanced, and the rules must include explicit move actions to advance □ and mark actions to keep it from returning to previously visited nodes.

With this notation for conditions and actions, the APSG rule that defines S may be stated. The condition for invoking the S rule is that the current relation node ○ must be of type AGNT:

```
S(type(○) is AGNT)→
   NP (move AGNT→□; mark AGNT→□ traversed;
     case := NOMINATIVE
     person := person of referent(□);
     number := number of referent(□))
   VP (move AGNT←□; voice :=ACTIVE;
     tense := tense of S; mode := mode of S;
     person := person of NP; number := number of NP).
```

On the right of the arrow, the actions for the NP record move the current concept node □ to the concept attached to the arc pointing away from the node ○, mark ○ traversed so that no other rule will traverse it again, set the case attribute NOMINATIVE (needed for pronouns, but not for nouns in English), and finally get the person and number attributes from □. The actions for VP move the node □ to the concept attached to the arc pointing towards ○, set the voice attribute ACTIVE, copy the tense and mode attributes from the original S node, and copy the person and number attributes from the record for the preceding NP node.

If sentence generation had started at a relation of type other than AGNT, the preceding rule would not apply. In that case, the system would search for another rule. If the starting relation had been of type OBJ, it would generate a sentence in passive voice:

```
S(type(○) is OBJ)→
   NP (move OBJ→□; mark OBJ→□ traversed;
     case := NOMINATIVE;
     person := person of referent(□);
     number := number of referent(□);
   VP (move OBJ←□; voice := PASSIVE;
        tense := tense of S; mode := mode of S;
        person := person of NP; number := number of NP).
```

The form of this rule is identical to the previous one, but the voice attribute of the VP record is now set to PASSIVE, and the NP node is generated from the object of the action instead of the agent. Other rules that apply to a VP record in passive voice will generate a form of the verb *be* and the preposition *by* for the agent (if it is present in the graph).

Unlike transformational grammar, APSG rules need no global movement operations that transpose subject and object. When sentence generation is started at OBJ instead of AGNT, the rules methodically move from node to node in the conceptual graph and end up with a globally well-formed passive sentence. At no time do the rules ever consider anything but local information in nearby nodes of a conceptual graph or records in the current APSG rule. The following rule expands a VP in passive voice in order to generate a form of the verb *be* followed by the main verb as a past participle:

```
VP (voice is PASSIVE)→
   VERB (type := BE; tense := tense of VP; mode := mode of VP;
     person := person of VP; number := number of VP)
   VP (form := PASTPART).
```

The VERB record has the verb type set to BE; the tense, mode, person, and number are copied from the original VP record on the left. The only attribute for the new VP record on the right of the arrow is the form attribute set to PASTPART. The new VP no longer has any information about tense, mode, person, or number. The type of the main verb is not stated in the record; the type will be copied from the current node of the conceptual graph when it is needed.

Passive verb phrases can be generated either for main verbs in passive voice or for participial phrases modifying nouns, as in the following sentences:

The books were distributed by the teacher.
Norma ordered books distributed by the teacher.

In both sentences, the phrase, *distributed by the teacher*, has exactly the same form and, for economy, should be generated by exactly the same grammar rules. Following is an APSG rule for noun phrases modified by past participial phrases:

```
NP (□←OBJ is not traversed)→
   NP (mark OBJ traversed; case := case of NP)
   VP (move OBJ←□; form := PASTPART).
```

Transformational grammar generates participial phrases by deletions from relative clauses. Yet clauses are more cumbersome, complex constructions than participial phrases. With APSG rules, participial phrases and infinitives can be generated directly by the simpler rules, and clauses are generated as special options for greater emphasis.

The AGNT relation is expressed by the preposition *by* in passive verb phrases, but it is not expressed by any special morpheme in a simple active sentence. The following rule generates a prepositional phase for the agent in passive form:

```
VP (form is PASTPART; □→AGNT is not traversed)→
   VP (form := PASTPART; mark AGNT traversed)
   PP (type := BY; move AGNT→□).
```

The concept [BOOK: {*}], which is uttered as *books*, is the object of both [ORDER] and [DISTRIBUTE] in the following graph:

```
[PEARSON: Normal]→(AGNT)→[ORDER]-
   (OBJ)→[BOOK: {*}]←(OBJ)←(DISTRIBUTE]←(AGNT)←[TEACHER: #8197].
```

If the utterance path starts at the AGNT relation attached to [ORDER], APSG rules generate the sentence,

Norma ordered books distributed by the teacher.

If the path had started at the AGNT relation linked to [DISTRIBUTE], the same rules would generate the sentence,

The teacher distributed books ordered by Norma.

Verb phrases may also include adverbs, direct objects, indirect objects and prepositional phrases. Following is a rule that generates adverbs:

```
VP (□→MANR is not traversed)→
    ADV (move MANR→□; mark MANR traversed)
    VP (vform := vform of VP).
```

To simplify the amount of copying that must be done, attributes may be grouped: vform is the collection of tense, mode, person and number that may be copied with a single assignment. The next rule generates direct objects:

```
VP (□→OBJ is not traversed)→
  VP (vform := vform of VP)
  NP (move OBJ→□; mark OBJ traversed;
      case := OBJECTIVE).
```

After all of the conceptual relations attached to the concept node corresponding to the main verb have been processed, the following rule generates the verb itself:

```
VP→VERB(type := type(□); vform := vform of VP).
```

In determining which rule to execute next, the system performs the first rule for the current nonterminal symbol for which the conditions are true. The simple rule that defines VP as just a VERB should therefore be the last one in the list for VP; since it has no conditions, it will always succeed, and no subsequent VP rule would ever be performed.

To generate noun phrases, the next rule defines NP as a sequence of determiner, adjective, and noun:

```
NP (referent(□) not proper name; □→ATTR not traversed)→
  DET (get referent from □)
  ADJ (move ATTR→□; mark ATTR traversed)
  NOUN (get type, number from □).
```

This rule would only apply if the referent of the current node □ was not a proper name and if the node □ was linked to a relation of type ATTR. The action associated with DET extracts information from the referent of □, the action for ADJ moves along the utterance path to the node on the other side of the ATTR relation, and the action for NOUN extracts the type and the number of the referent from the node □.

After the phrase-structure rules have been applied, lower-level, *lexical rules* must be used to generate the actual words of the language. The lexical rules take into account the concept type, the syntactic category, and other attributes of the nonterminal symbol. What they generate is a character string (or in the case of spoken language, a phonemic representation):

```
NOUN (type is BABY; number is PLURAL)→"babies".
NOUN (type is BABY; number is SINGULAR)→"baby".
VERB (type is DRINK; tense is PAST)→"drank".
```

The same concept type may be mapped into different word forms for different syntactic categories:

> NOUN (type is DISTRIBUTE; number is SINGULAR)→"distribution".
> VERB (type is DISTRIBUTE; number is PLURAL; tense is PRESENT)→
>   "distribute".

In principle, a separate lexical rule may be stated for every word form. In practice, however, a unified morphological stage would look up each concept type in a dictionary and generate the appropriate word for it. The morphological stage could take account of regular rules like *−s* for most plural nouns and would only need special rules for exceptions. It could be a much simpler routine than the APSG processor since it would only have to consider adjacent words in order to generate the article *a* before a consonant or *an* before a vowel.

Generating correct articles in English is difficult because the words *the* and *a* have many different uses. At the first mention of an object, the indefinite article *a* introduces it, but subsequent references use the definite article *the* to refer back to it. Often, however, *the* is used to refer to an object that is implicitly introduced into the context:

> Do you have a 1982 penny? I want to check the weight.

Although the weight of the coin was not explicitly mentioned, all of the usual attributes of an object may be assumed as part of the current context whenever the object itself is introduced. Besides the use of articles for individuals, both *the* and *a* may be used in a generic sense:

> The horse is a noble animal.
>
> A dog is an animal.

A complete study of English articles would require a major treatise. As illustrations, the following three APSG rules generate *the* if the concept is individual, generate *a* if it is generic, and generate the empty string ϵ if it is a generic set (plural):

> DET (*referent*(□) is individual)→"the".
> DET (*referent*(□) is generic)→"a".
> DET (*referent*(□) is {*})→ϵ.

The conditions inside the parentheses could be elaborated to handle other versions of English grammar that make finer distinctions.

Prepositions are usually generated from conceptual relations rather than concept nodes. They are especially sensitive to the direction of the utterance path. Consider the following subgraph of Fig. 2:

> [BABY: {*}]→(PART)→[BELLY: {*}].

If the utterance path is moving from [BABY] to [BELLY], the resulting phrase would be *babies with bellies*. But if it is moving from [BELLY] to [BABY], the result would be *bellies of babies*. Unlike lexical rules for nouns and verbs, the rules for generating prepositions depend on the direction of the utterance path. Following is a rule for generating prepositional phrases:

```
PP→
    PREP (copy type, direction from PP)
    NP (set case OBJECTIVE).
```

Since there are no conditions on the left, this rule applies whenever any prepositional phrase is being generated. Since there is no move stated in the action lists, both PREP and NP have the same □ node as PP. The type and direction are copied from the PP record to the PREP record.

and the NP record has the case attribute set to OBJECTIVE. In English, the case is needed for pronouns, but it is ignored for nouns; other languages would set different case attributes for NP depending on the relation type. The lexical rules for prepositions come in pairs depending on the direction of the utterance path. Following are the rules for the PART and CONT (content) relations:

PREP (type is PART; direction is→)→"of".
PREP (type is PART; direction is ←)→"with".
PREP (type is CONT; direction is →)→ "with".
PREP (type is CONT; direction is ←)→"in".

Note that there is no one-to-one mapping between relations and prepositions: the preposition *with* could have been generated for the types PART and CONT in these rules as well as for INST (instrument) and ACCM (accompaniment).

One of Fillmore's principles of case grammar[19] is that the agent if present becomes the subject; if the agent is missing, then the next choice for subject is the instrument. In APSG rules, that principle would be implemented with a rule like the following:

S(type(O) is INST)→
  NP (move INST→□; mark INST traversed; get person and number from □)
  VP (move INST←□; set voice ACTIVE; copy person and number from NP).

If the earlier rule for S failed, then this would be the next one tried. If these conditions also failed, then the system would continue to scan the list of rules for S until it found one whose conditions were satisfied. At the end of the list, there could be a default rule like the following that would print an error message:

S→ "Sorry, I don't know how to say what I mean."

Since this rule has no conditions, it will always succeed if invoked. But since it isn't very helpful, it is a last resort.

To generate a sentence in Japanese, the following rule could be used to generate the SOV word order. The rule also sets attributes of the noun phrases to generate the postpositions *ga* and *o*, which indicate subject and object in Japanese:

S (AGNT and OBJ links present & not yet traversed)→
  NP (select node linked to AGNT, set postposition GA)
  NP (select node linked to OBJ, set postposition O)
  VERB (mark AGNT an OBJ links traversed).

Appropriate changes in the rules could generate the word orders for any of the other languages, such as Hebrew or French. Type labels like BIRD and BABY happen to look like English words, but they represent abstract concepts. For generating French, lexical rules like the following could be used:

NOUN (type is BIRD; number is SINGULAR)→"oiseau".
NOUN (type is BIRD; number is PLURAL)→"oiseaux".

Although the APSG rules described in this section have never been implemented in exactly the form described here, they are adapted from the encoding rules in Heidorn's NLP system and could be mapped into those rules on a one-for-one basis. The primary difference is that these APSG rules have been specialized to the notation for conceptual graphs, but NLP rules can process other kinds of graphs as well. The following passage was generated by Heidorn's NLPQ system[16] using APSG rules similar to the ones described in this paper:

The vehicles arrive at the station. The time between arrivals of the vehicles at the station is normally distributed, with a mean of 8 min and a standard deviation of 2 min. Seventy-five percent of the vehicles are cars, and the rest are trucks. After arriving at the station, if the length of the line at the pump in the station is less than 2, the vehicles will be serviced at the pump in the station. Otherwise, the vehicles will leave the station. The time for the vehicles to be serviced at the pump in the station is exponentially distributed, with a mean of 5 min for the cars, and 9 min for the trucks. After being serviced at the pump in the station, the vehicles leave the station. (p. 5).

The same graph structure that was mapped into this English passage could also be mapped into other languages—computer languages as well as natural languages. For NLPQ, Heidorn wrote two sets of encoding rules. One set produced the English paragraph above, and the other set mapped the graphs into a program in the GPSS simulation language.

4. MAPPING GRAMMAR, RATHER THAN GENERATIVE GRAMMAR

Speaking involves three stages: determining what to say, how to relate it to the listener, and how to map it into a string of words. In terms of conceptual graphs, the first stage is the selection of some graph or collection of graphs to be expressed. The second stage uses pragmatic concerns to select the starting point and direction of the utterance path, which determines which concept will be the subject and which will be the main predicate. The third stage scans the graph and maps concepts into words. Determining what to say and how to relate it to the listener involves issues of inference and pragmatics that are treated in Sowa[8]; this article has presented the third stage of using APSG rules to map a graph to a sentence.

In generating sentences, APSG rules are invoked in a top-down, goal-directed fashion. The algorithm starts with a single goal—generate sentence, generate paragraph, or even generate story. The initial goal contains a pointer to some node of a conceptual graph: if the goal is to generate a sentence, the pointer would usually select a concept corresponding to the main verb; if the goal is to generate a paragraph or story, the pointer might select a time-ordered sequence of actions, each of which represents a single sentence. The rule invoked for the initial goal makes some tests on the conceptual graph and generates other subgoals —generate noun phrase or generate verb phrase—each of which makes its own tests and generates further subgoals down to the lowest level goals like generating a present tense, third-person, singular form to *be*. As each rule invokes other rules as subgoals, the APSG processor makes a cyclic walk of the conceptual graph: the goal of generating a sentence starts at the relation between subject and main predicate, that goal generates a subgoal that follows the AGNT link to the subject, which may in turn generate further subgoals that follow links for adjectives and other modifiers. When each subgoal is finished, the walk returns to the node that invoked it. At the end, the highest-level subgoal returns to the topmost goal, which corresponds to the starting node of the conceptual graph.

Unlike parsing programs, which use many different techniques, most language generators are based on top-down algorithms, but with different notations and terminology. Simmons and Slocum[20] and Wong[21] use *augmented transition nets*, which are also executed in a top-down fashion. Wong took care to generate correct anaphoric expressions: when introducing an event, his program would say, for example, *A boy broke a window*; but when referring back to the boy, it would say *he*, *the boy*, or *the boy who broke the window*, depending on the amount of detail needed to specify the referent uniquely. Goldman's BABEL[22] is a top-down program for mapping Schank's conceptual dependency graphs into English. One of Goldman's innovations was to make word choices based on *word-sense discrimination nets*. Since Schank's theory now permits high-level concept types like ADMIT and THREATEN, the major discriminations could be made by an earlier inference stage. BABEL, however, had to make all the word choices itself since its input graphs contained only low-level primitives. Although these authors do not use the term *utterance path* and they do not use the APSG notation, at an abstract level, their algorithms are similar: they execute rules in a top-down order, the graph serves as a control for determining which rules to select, and the order of processing nodes may be described as an utterance path.

Although APSG rules have an underlying skeleton that resembles Chomsky's phrase-structure rules, it is more accurate to call them a mapping grammar, rather than a generative grammar. The theory presented in this article meets Chomsky's original goals of relating multiple surface structures to a common deep structure, but with some major advantages:

• Psychological naturalness: Generation is done by rules of inference and pragmatics, which deal with meaning, rather than rules of syntax that only deal with the forms of expression.

• Computational efficiency: Unlike transformational rules that move large constituents during the generation stage, APSG rules obtain the effects of transformations by changing the direction of the utterance path rather than by moving words and phrases. Since the rules generate words one at a time as they are uttered, they reduce the demands either on human short-term memory or on computer storage.

• Theoretical simplicity: Constraints on transformations arise from the connectivity of the graphs and the possible ways of scanning them. No special assumptions are needed to block transformations that never occur in natural languages because those transformations violate the universal rules that govern the utterance path.

To give a detailed comparison of the conceptual graph theory with transformational grammar is beyond the scope of this article. But the notion of *traces*[23] illustrates the difference between the two approaches: a trace is a residue of a noun phrase that has been moved by a transformation; it has no sound itself, but it changes the intonation pattern of the sentence and blocks certain kinds of contractions. The following example shows the trace *t* that is left when *what* is moved to the front of a sentence by the question transformation:

Ruby gave what to Clara?
What did Ruby give *t* to Clara?

In terms of Chomsky's approach, a trace is like a "silent pronoun" that is left behind when a noun phrase is moved from one position in a sentence to another. In terms of conceptual graphs, such fictitious pronouns are not needed: the point at which a trace occurs is a point where the utterance path visits a concept whose utterance mark is set on. The blocking of contractions is caused by a break in normal processing when the utterance path stops at a node and bypasses it. As this example illustrates, conceptual graphs can explain the same kinds of phenomena as transformational grammar, but with fewer *ad hoc* assumptions and a less cumbersome formalism.

Acknowledgements—I would like to thank George Heidorn and Karen Jensen for a number of comments and suggestions that have helped to improve the content and presentation of this paper.

REFERENCES

1. C. S. Peirce, Unpublished manuscripts summarized in D. D. Roberts, *The Existential Graphs of Charles S. Peirce*. Mouton, The Hague (1973).
2. L. Tesnière, *Elements de Syntaxe Structurale*, 2nd Edn. Librairie C. Klincksieck, Paris (1965).
3. S. Ceccato, *Linguistic Analysis and Programming for Mechanical Translation*. Gordon & Breach, New York (1961).
4. M. Masterman, Semantic message detection for machine translation, using an interlingua. *Proc.* 1961 *Int. Conf. on Machine Translation*, pp. 438–475 (1961).
5. R. C. Schank and G. Tesler, A conceptual parser for natural language. *Proc. IJCAI*-69, pp. 569–578 (1969).
6. G. G. Hendrix, Expanding the utility of semantic networks through partitioning. In *Proc. IJCAI*-75, pp. 115–121 (1975).
7. R. J. Brachman, On the epistemological status of semantic networks. In *Associative Networks: Representation and Use of Knowledge by Computers* (Edited by N. V. Findler), pp. 3–50. Academic Press, New York (1979).
8. J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, Mass. (1983).
9. M. R. Quillian, Semantic memory. *Rep. AD*-641671, Clearinghouse for Federal Scientific and Technical Information.
10. D. McNeill, *The Conceptual Basis of Language*. Lawrence Erlbaum, Hillsdale, New Jersey (1979).
11. J. H. Greenberg, Some universals of grammar with particular reference to the order of meaningful elements. In *Universals of Language* (Edited by J. H. Greenberg), pp. 58–90. MIT Press, Cambridge, Mass (1963).
12. S. Steele, Word order variation: a typological study. In *Universals of Human Language* (Edited by J. H. Greenbrg), 4 vols., pp. 585–623. Stanford University Press, Stanford, Calif. (1978).
13. N. Chomsky, *Syntactic Structures*. Mouton & Co., The Hague (1957).
14. J. F. Sowa, Conceptual structures: A model for language. Unpublished manuscript (1968).
15. P. M. Postal, *On Raising*. MIT Press, Cambridge, Mass. (1974).
16. G. E. Heidorn, Natural language inputs to a simulation programming system. *Rep. NPS-55HD72101A*, Naval Postgraduate School, Monterey (1972).

17. G. E. Heidorn, Augmented phrase structure grammar. In *Theoretical Issues in Natural Language Processing* (Edited by R. Schank and B. L. Nash-Webber) pp. 1-5.
18. D. E. Knuth, Semantics of context-free languages. *Math. Systems Theory* **2**, 127-145.
19. C. J. Fillmore, The case for case. In *Universals in Linguistic Theory* (Edited by E. Bach and R. T. Harms), pp. 1-88. Holt, Rinehart & Winston, New York (1968).
20. R. F. Simmons and J. Slocum, Generating English discourse from semantic networks. *Commun. ACM* **15**, 891-905 (1972).
21. H. K. T. Wong, Generating English Sentences from Semantic Structures. *Tech. Rep.* 84, Dept of Computer Science, University of Toronto (1975).
22. N. M. Goldman, Conceptual generation. In *Conceptual Information Processing* (Edited by R. C. Schank) pp. 289-371. North-Holland, Amsterdam (1975).
23. N. Chomsky, Conditions on rules of grammar. In *Current Issues in Linguistic Theory* (Edited by R. W. Cole), pp. 3-50. Indiana University Press, Bloomington (1977).