INTERPRETATION IN GENERATION

Eduard H. Hovy
Information Sciences Institute of USC[1]
4676 Admiralty Way
Marina del Rey, CA 90292-6695
Telephone: 213-822-1511
HOVY @ VAXA.ISI.EDU

Abstract

The computer maxim *garbage in, garbage out* is especially true of language generation. When a generator slavishly follows its input topics, it usually produces bad text. In order to find more appropriate forms of expression, generators must be given the ability to interpret their input topics. Often, newly formed interpretations can help generators achieve their pragmatic goals with respect to the hearer. Since interpretation requires inference, generators must exercise some control over the inference process. Some general strategies of control, and some specific techniques geared toward achieving pragmatic goals, are described here.

1 The Problem

Simply put, the generator's task, for a given sentence topic, is to find a form of expression — either a syntactic rule or a phrase — that will enable it to select and to order aspects of the topic in order to build a sentence. The straightforward approach is to define a fixed correspondence between topic representation types on the one hand and grammatical rules and lexical elements on the other. This approach has a flaw: the results are invariably bad or boring. How bad, of course, depends on the representation, but anything detailed enough to be useful for other purposes, such as learning or diagnosing, simply does not make great prose in practice. A good example is furnished by the following text, in which the generator's input consists of a list of topics, where each topic describes some episode in a fight between two people[2]. Straightforward generation produces:

(a) FIRST, JIM BUMPED MIKE ONCE, HURTING HIM. THEN MIKE HIT JIM, HURTING HIM. THEN JIM HIT MIKE ONCE, KNOCKING HIM DOWN. THEN MIKE HIT JIM SEVERAL TIMES, KNOCKING HIM DOWN. THEN JIM SLAPPED MIKE SEVERAL TIMES, HURTING HIM. THEN MIKE STABBED JIM. AS A RESULT, JIM DIED.

This example[3] is an extreme case because it contains only two main representation types, ACTION and STATE, which can relate in only one way, RESULT. When the generator knows only one way to express this combination, what more can we hope for?

Correcting this inflexibility seems straightforward. Though there is nothing wrong with the sentence form used above, namely,

[[*say-time* #TIME] [*say-sentence* #ACTION] , [*say-participle* #STATE]]

one can add to the grammar a few more sentence forms expressing actions and their results, more time words, and more verbs, and then make the generator cycle through its options whenever it encounters a choice point:

(b) FIRST, JIM BUMPED MIKE ONCE AND HURT HIM. THEN MIKE SMACKED JIM, HURTING HIM. NEXT, JIM HIT MIKE ONCE. THE RESULT WAS THAT HE KNOCKED HIM DOWN. AFTER THAT, MIKE SMACKED JIM SEVERAL TIMES AND KNOCKED HIM DOWN. JIM SLAPPED MIKE SEVERAL TIMES, HURTING HIM. AFTER THAT, MIKE STABBED JIM. AS A RESULT, JIM DIED.

Yet this produces no real improvement! Clearly, simply extending the number of phrase patterns for each representation type does not solve the problem. When we

[1]This work was done while the author was at Yale University Computer Science Department, 2158 Yale Station, New Haven, CT 06520. This work was supported in part by DARPA monitored by the ONR under contract N00014-82-K-0149. It was also supported by AFOSR contract F49620-87-C-0005.

[2]The input was produced by the JUDGE program (see [Bain 86] and [Bain 84]), a case-based expert system that models the sentencing behavior of a judge. As input, JUDGE accepts the representation of a fight — a set of actions and resulting states — and as output it produces a set of interpretations of each action.

[3]All the texts in this paper were generated by PAULINE (Planning And Uttering Language In Natural Environments), a program that can realize a given input in a number of different ways, depending on how its pragmatic goals are set. An overview description can be found in [Hovy 87a, 87b]. The program consists of over 12,000 lines of T, a Scheme-like dialect of LISP developed at Yale.

speak, we do a lot more than simply cast input topics in different forms; for example, we might say:

(c) JIM DIED IN A FIGHT WITH MIKE.

(d) AFTER JIM BUMPED MIKE ONCE, THEY FOUGHT, AND EVENTUALLY MIKE KILLED JIM.

(e) AFTER JIM BUMPED MIKE ONCE, THEY FOUGHT, AND EVENTUALLY HE WAS KNOCKED TO THE GROUND BY MIKE. HE SLAPPED MIKE A FEW TIMES. THEN MIKE STABBED JIM, AND JIM DIED.

Illustrated this way, the problem seems rather simple. Obviously, the solution is to group together similar enough topics, where the similarity criterion can be varied depending on external factors, and then to generate the groupings instead of the individual actions. Grouping together contiguous actions of similar force, PAULINE produced variants (c), (d), and (e). (In the first variant, all actions were grouped together; in the second, all actions more violent than bumping but less violent than killing were accepted; and in the third, the grouping resulted from defining four levels of violence: bumping, hitting and slapping, knocking to the ground, and killing.)

Clearly, though it improves the JUDGE examples, the technique of grouping actions by levels of force is very specific and not very useful. However, when "group" is used in a wider sense to mean "interpret", this technique becomes both difficult and interesting, and provides a very powerful way to increase the expressive flexibility and text quality of a generator. So the questions are: what interpretation/grouping criteria are general and still useful? When and how should the generator interpret input topics? How should it find appropriate grouping criteria?

2 An Example of Interpretation

In a second example, PAULINE produces a number of versions describing a hypothetical primary election between Carter and Kennedy during the 1980 Democratic Presidential nomination race. In thi election, Kennedy narrows Carter's lead[4]. When PAULINE is given as input the outcome for each candidate, straightforward generation produces:

(f) IN THE PRIMARY ON 20 FEBRUARY CARTER GOT 1850 VOTES. KENNEDY GOT 2185.

[4]This event is represented using about 80 elements of a representation scheme similar to Conceptual Dependency [Schank 72, 75, 82], defined in a property-inheritance network such as described in [Charniak, Riesbeck & McDermott 80].

However, PAULINE can notice that both outcomes relate to the same primary, and can say instead:

(g) IN THE PRIMARY ON 20 FEBRUARY, KENNEDY BEAT CARTER BY 335 VOTES.

(or any of a number of similar sentences using "beat", "win", and "lose"). But why stop there? If PAULINE examines the input further, it can notice that Carter's current delegate count is greater than Kennedy's, that this was also the case before the primary, and that this primary is part of a series that culminates in the final election, the nomination. In other words, PAULINE can recognize that what happened in this primary was:

(h) IN THE PRIMARY ON 20 FEBRUARY, KENNEDY NARROWED CARTER'S LEAD BY GETTING 2185 VOTES TO HIS 1850.

If we want good text from our generators, we have to give them the ability to recognize that "beat" or "lose" or "narrow lead" can be used instead of only the straightforward sentences (f).

This ability is more than a simple grouping of the two outcomes. It is an act of generator-directed inference, of interpretation, forming out of the two topics a new topic, perhaps one that does not even exist in memory yet. And the new topic is not simply a generator construct, but is a valid concept in memory. The act of determining that "beat" is appropriate *is* the act of interpreting the input as an instance of BEAT — denying this is to imply that "beat" can logically be used where BEAT is not appropriate, which is a contradiction. This is not an obvious point; one could hold that the task of finding "beat" to satisfy a syntactic or pragmatic goal is a legitimate generator function, whereas the task of instantiating it and incorporating it into memory is not. However, it is clearly inefficient for a generator to interpret its input, say it, and then simply forget it again! — especially when there is no principled reason why generator inferences should be distinct from other memory processes.

Thus, after interpretation, the newly built instance of the concept should be added to the story representation, where it can also be used by other processes, or by the generator the next time it tells the story. In this way the content of memory can change as a result of generation. This is consistent with the fact that you often understand a topic better after you have told someone about it: the act of generating has caused you to make explicit and to remember some information you didn't have before.

Immediately, this view poses the question: *which process is responsible for making these inferences?*. The two

possible positions on this issue reflect the amount of work one expects the generator to do. According to the strict minimalist position — a position held by most, if not all, generator builders today —, the generator's responsibility is to produce text that faithfully mirrors the input topics with minimal deviation: each sentence-level input topic produces a distinct output sentence (though perhaps conjoined with or subordinated to another). Naturally, this inflexible attitude gave rise to the JUDGE texts (a) and (b). To circumvent this problem, in practice, most generator builders employ in their programs a number of special-purpose techniques, such as sophisticated sentence specialists that are sensitive to the subsequent input topics. Of course, this is a tacit acknowledgement that the strict position does not hold. However, on renouncing the hard-line position, one must face the question *how much generator-directed inference are you prepared to do?*

I do not believe that a simple answer can be given to this question. The issue here, I think, is economic: a tradeoff exists between the time and effort required to do interpretation (which includes finding candidate interpretations, making them, and deciding on one) on the one hand, and the importance of flowing, good text on the other. Greater expense in time and effort produces better text. Thus pragmatic criteria are appropriate for treating this question. Hence a reasonable answer is *I'll do as much inference as I can do, given the available time, the pragmatic constraints on what I want the hearer to know, and the richness of my memory and my lexicon.* Of these three factors, the most difficult is clearly the pragmatic constraints on what the hearer is to be told. When does the hearer need to know the details of the topic? What is the effect of telling him only interpretations? Or of telling him both? The answer can be summarized as: if you can trust him to make the interpretations himself, then all you need give him are the details. Thus, if the hearer is a political pundit who is following the nomination race with interest, then clearly (f) is better, since he can draw the conclusion without difficulty, and, in addition, he has precise numerical information. If, in contrast, the hearer has only minimal knowledge about or interest in the nomination procedure, then (h) is better, since it doesn't burden him with details and require him to do the interpretation himself. What must you say, however, if the hearer is interested and has a limited amount of knowledge — say, he is a student of the political process —, or if he is knowledgeable but unlikely to make the right interpretation — say, he is a strong Kennedy supporter, whereas you are pro-Carter? In both these cases you must ensure that the hearer understands how you expect him to interpret the facts. So you tell him details *and* the interpretations:

(i) KENNEDY NARROWED CARTER'S LEAD IN THE PRIMARY ON 20 FEBRUARY. HE GOT 2185 VOTES AND CARTER GOT 1850.

In summary, you must be as specific as the hearer's knowledge of the topic allows: if you are too specific he won't understand, and if you are too general you run the risk of seeming to hide things from him, or of being uncooperative. In the first case, you violate the goal to be intelligible, and in the second, you violate the goal to avoid unacceptable implications. In either case, you violate Grice's maxim of quantity to say neither more nor less than is required (see [Grice 75]).

3 Where Do Candidate Interpretations Come From?

The problem in interpretation is to find valid interpretations easily and quickly.

Bottom-Up Interpretation: One solution to this problem is to try inferences directly on the input topics. This bottom-up method of interpretation uses the structure of the memory network itself.

In PAULINE, bottom-up interpretation inferences reside in memory and the lexicon as part of the definitions of concept types. In order to enable bottom-up interpretations, links are defined from concept types to the interpretations in which they could take part. (This scheme forms a concept representation network slightly different from the usual multi-parent schemes used in, say, [Stefik & Bobrow 85], [Charniak, Riesbeck & McDermott 80], and [Bobrow & Winograd 77].) Of course, this is not a wonderful solution — it depends on the right links being defined beforehand — but it is practical in limited domains. The program collects possible inferences from the type of each input topic.

Top-Down Interpretation: Another way to find interpretations is top-down: to run only the inferences likely to produce results that serve the generator's pragmatic goals. Potentially useful inferences can be explicitly included in the plans, and can be tried on candidate sentence topics whenever they are collected. Since interpretation is a powerful way of slanting the text, the pragmatic goals to communicate opinions are an eminently suitable source of guidance[5]. Indeed, many of these goals can *only* be achieved through interpreting the input topics appropriately.

[5]How PAULINE is given opinions and some of its techniques for slanting are described in [Hovy 86].

PAULINE's strategies for slanting its text include a number of top-down interpretation inferences. For example, one strategy for an unsympathetic action is

> **Interpret as confrontation:** state that the actor you oppose (X) did some action (ACT) as a confrontation with some actor you support (Y). This rule can be represented as:

```
IF X has the goal that some actor B must
   do some action C
   AND Y has goal that B must do C'
   AND C' conflicts with C
   AND X's action ACT forces B to do C'
       (disregarding Y)
THEN interpret ACT as a confrontation
```

4 How PAULINE Does It

In order to interpret the input topics as instances of some concept, the interpretation process must recognize when the topics (or some of them) conform to the definition (or part of the definition) of the concept. Thus, either concepts must be defined in such a way as to allow a general process to read their definitions, or inferences must exist that fire when a definition is matched — in other words, the antecedent of an inference is the definition and the consequent asserts the existence of the new concept.

PAULINE was implemented with the second approach, using patterns called configurations. A configuration is the description of the way in which a collection of concepts must relate to one other to form a legitimate instance of a high-level concept. It contains a pattern, in the form of a list of triplets (*type ?var pattern*), where

- *type* is either the type (in the property inheritance memory network) of the concept currently to be matched, or a variable *?var* which must have been encountered before.
- *?var* is either (), or a variable *?var* by which the current concept will be identified later in the match, or two such variables that have to be bound to different concepts for a match.
- *pattern* is a list of (*aspect config*) pairs, where the filler of each *aspect* must recursively match the *config*, which is again a pattern.

Configuration patterns obviously depend on the exact representations used. For example, the configuration for the concept BEAT is

```
; ?X is someone's VOTE-OUTCOME
; in some primary ?Y,
  (VOTE-OUTCOME ?X
    (instance (ELECTION ?Y))
    (relations (REL-GREATER ()
; and it is greater than
; another VOTE-OUTCOME in ?Y
                  (conc1 (?X))
                  (conc2 (VOTE-OUTCOME ()
                           (instance (?Y)))))))
```

which means: some concept is a VOTE-OUTCOME; its aspect RELATIONS contains a GREATER relation, of which the greater part is that same concept and the smaller part is another VOTE-OUTCOME in the same primary. Thus, since Kennedy's outcome resulted from a primary and it is greater than Carter's outcome, the two form an instance of BEATing. Most configurations are considerably more complex.

During its planning stage, PAULINE gathers likely interpretation inferences, both top-down and bottom-up, and then, using a simple pattern-matcher, applies their configurations to the candidate topics and collects all the matches. Its strategies for selecting configurations are based upon the pragmatic factors knowledge, slant, and time, described above. If an instance of a newly made interpretation does not yet exist in memory, PAULINE creates one and indexes it following the memory organization principles described in [Schank 82], so that it can be found again and used in future.

A final example: PAULINE generates over 100 versions (neutral, slanted in various ways, in various styles) of an episode that took place at Yale in April 1986. The episode requires about 120 representation elements. A neutral version is:

> (j) IN EARLY APRIL, A NUMBER OF STUDENTS BUILT A SHANTYTOWN ON BEINECKE PLAZA. THE STUDENTS WANTED YALE UNIVERSITY TO DIVEST FROM COMPANIES DOING BUSINESS IN SOUTH AFRICA. ONE MORNING, OFFICIALS DESTROYED THE SHANTYTOWN AND POLICE ARRESTED 76 STUDENTS. FINALLY, THE UNIVERSITY ALLOWED THE STUDENTS TO REBUILD IT, AND ANNOUNCED THAT A COMMISSION WOULD GO TO SOUTH AFRICA IN JULY TO EXAMINE THE SYSTEM OF APARTHEID.

When generating with the goal to slant the input, PAULINE uses top-down inferences such as those mentioned above to interpret the input topics. For example, an anti-protester text is:

(k) IN EARLY APRIL, A SMALL NUMBER OF STUDENTS [WERE INVOLVED IN A CONFRONTATION]$_a$ WITH YALE UNIVERSITY OVER YALE'S INVESTMENT IN COMPANIES DOING BUSINESS IN SOUTH AFRICA. THE STUDENTS [TOOK OVER]$_b$ BEINECKE PLAZA AND CONSTRUCTED A SHANTYTOWN NAMED WINNIE MANDELA CITY [IN ORDER TO FORCE]$_c$ THE UNIVERSITY TO DIVEST FROM THOSE COMPANIES. YALE REQUESTED THAT THE STUDENTS ERECT IT ELSEWHERE, BUT THEY REFUSED TO LEAVE. LATER, AT 5:30 AM ON APRIL 14, OFFICIALS HAD TO DISASSEMBLE THE SHANTYTOWN. FINALLY, YALE, [BEING CONCILIATORY]$_d$ TOWARD THE STUDENTS, NOT ONLY PERMITTED THEM TO RECONSTRUCT IT, BUT ALSO ANNOUNCED THAT A COMMISSION WOULD GO TO SOUTH AFRICA IN JULY TO EXAMINE THE SYSTEM OF APARTHEID.

PAULINE made the interpretations *confrontation* (a), *appropriation* (b), *coercion* (c), and *conciliation* (d), none of which were contained in the original input story.

5 Conclusion

As generators become larger and more complex, and as they are increasingly used together with other programs, they should use the capabilities of those programs to further their own ends. Therefore, we should study the kinds of tasks that generators share with other processes and the purposes generators require them to fulfill. The considerations and strategies described here determine some of the kinds of demands a generator can be expected to place on a general-purpose inference engine. And even with PAULINE's limited inferential capability, the program can greatly enhance the quality of its text and the efficiency of its communication of non-literal pragmatic information.

6 References

1. Bain, W.M.,
Toward a Model of Subjective Interpretation, Yale University Technical Report no 324, 1984.

2. Bain, W.M.,
Case-based Reasoning: A Computer Model of Subjective Assessment, Ph.D. dissertation, Yale, 1985.

3. Bobrow, D.G. & Winograd, T.,
An Overview of KRL, a Knowledge-Representation Language, in *Cognitive Science* vol 1 no 1, 1977.

4. Charniak, E., Riesbeck, C.K. & McDermott, D.V.,
Artificial Intelligence Programming, Lawrence Erlbaum Associates, 1980.

5. Grice, H.P.,
Logic and Conversation, in **The Logic of Grammar**, Davidson D. & Harman G. (eds), Dickinson Publishing Company, 1975.

6. Hovy, E.H.,
Integrating Text Planning and Production in Generation, IJCAI Conference Proceedings, 1985.

7. Hovy, E.H.,
Putting Affect into Text, Proceedings of the Eighth Conference of the Cognitive Science Society, 1986.

8. Hovy, E.H., 1987a,
Some Pragmatic Decision Criteria in Generation, in **Natural Language Generation: Recent Advances in Artificial Intelligence, Psychology, and Linguistics**, Kempen G. (ed), Kluwer Academic Publishers, 1987.

9. Hovy, E.H., 1987b,
Generating Natural Language under Pragmatic Constraints, in *Journal of Pragmatics*, vol XI no 6, 1987, forthcoming.

10. Schank, R.C.,
'Semantics' in Conceptual Analysis, in *Lingua* vol 30 no 2, 1972, North-Holland Publishing Company.

11. Schank, R.C.,
Conceptual Information Processing, North-Holland Publishing Company, 1975.

12. Schank, R.C.,
Dynamic Memory: A Theory of Reminding and Learning in Computers and People, Cambridge University Press, 1982.

13. Stefik, M. & Bobrow, D.G.,
Object-Oriented Programming: Themes and Variations, in *AI Magazine* Vol 6, No 4, 1986.