no. 209

NOMAD: A Naval Message Understanding System

Richard H. Granger
Amnon Meyers
Gregory B. Taylor
Rika Yoshii

Artificial Intelligence Project

Technical Report 209

April 26, 1983

ABSTRACT

We are building systems to automatically analyze Navy messages. Such messages typically are terse and use many abbreviations and Navy jargon. As a result, they are more difficult to understand than everyday English.

The NOMAD system interacts with a message sender to ensure that only unambiguous and reasonably correct messages are generated. The VOX system will allow a human tutor to interactively extend the knowledge base of NOMAD.

This research is supported by the Naval Ocean Systems Center under contract N00123-81-C-1078.

INTRODUCTION

NOMAD (Naval Onboard Message Analyzer and Disambiguator) is a computer program for understanding Naval messages. While most message understanding systems process communications at the receiving end, NOMAD interacts with the message sender to produce a well-formed message.

The Navy must process thousands of communications among ships, planes, subs, and bases every day. Naval messages as a rule are terse, ambiguous and ungrammatical. In order to comprehend these messages and convert them to database-readable form, the Navy is turning to automatic methods.

In this paper, we will describe the Navy domain and its problems. Then we will discuss two programs that approach the task of understanding Navy English.

NAVY DOMAIN

Typical Naval scenarios revolve around events at sea. Information about military operations, battles, maneuvers, communications, orders, and reports provides most of the substance of Naval messages. Events taking place in the air, on the land and sea surface, and under water are described.

Detailed knowledge of armaments also is necessary. It is important to have knowledge of ships, their capabilities, their interactions with other ships and with aircraft and the weapons available to them in order to understand the Navy domain.

Several works describe parts of the knowledge domain of the Navy [Bechtel, Moore].

Programs specializing in the Navy domain need not fully understand the complexities of human needs, goals and plans. The main goals center around maximizing enemy losses and minimizing losses among allies.

NAVY ENGLISH

The language of Navy messages bears little resemblance to everyday English. It contains many errors at the word and phrase level. Many abbreviations and technical jargon are used, or even ad-libbed. Many messages also have multiple meanings.

The writers of Navy messages seem to use as few words as possible to describe a situation. Connecting words like "the", "a", "an", "and" rarely appear. The subject and object of a sentence are often omitted. For example, PERISCOPE SIGHTED.

CONTINUING EFFORT TO LOCATE ALL ORANGE SURF UNITS
PRIOR DARKNESS.

Problems of ambiguity are made worse by the extreme terseness of Navy messages. For example,

CHALLENGED AGI REFUSED TO HEAVE TO.

might be interpreted in any of the following ways:

a) We challenged AGI. AGI refused to heave to.
b) We challenged AGI. We refused to heave to.
c) The challenged AGI refused to heave to.
d) When challenged, AGI refused to heave to.

Run-together sentences appear often. For example:

a) OPENED FIRE WITH 5IN 54 GUNS - 16 ROUNDS EXPENDED
CONDUCTED URGENT ATTACK WITH BLOODHOUND

b) SECOND ATTACK KOBCHIC RESPONDED WITH GUNS.

The first part of example (b) above is also a fragment containing no verb. Words as well as sentences are often merged incorrectly:

ENROUTE
UNDERFIRE
SPD5
CRASHCREW
BIRDSTRIKES

New words are also added to the language in the heat of Naval operations. For example, the word "limpeteer" was used in the phrase "limpeteer attack". "Limpet" refers to a mine which divers place under the hull of an enemy vessel, but "limpeteer" is not found in the Naval lexicon [Noel, for example]. Another new word is "oparea", which means "area of operations".

The Navy has a rich set of idioms and jargon:

a) TOOK KOBCHIC UNDER FIRE.
b) BOTH UNITS APPROACHED UNDER DARKEN SHIP.
c) BRIEF CONTACT HELD ON POSS SUB.
d) FANNING AND MILLER HAVE BROKEN ENGAGEMENT
AND RETURNING TO SCREEN CONCORD.

Following are some complete Navy messages:

NARR/LATE ONSTA DUE ACFT MALFUNCTIONS. UNABLE TO
CONTACT AAWC ON VIRGINIA. EST NESTER COMM WITH
ASWC AND RECEIVED SWAP INFO FROM OFFGOING P3.
TASS UNITS NOT TRAILING ARRAYS DUE HIGH SPEED OPS

AND SURFACE ENGAGEMENTS. MAJORITY ONSTA TASKING WAS AS COMM RELAY. UNDER DIRECTION OF ASWC LAYED 10 BUOY BRUSHTAC AT 2340Z. GAINED OP CTC. PASSED CTC VIA NESTOR SWAP TO RELIEF AND DEPARTED AREA AT 0100Z. CONTACT LATER EVAL AS SURF.

NARR/VISUAL SIGHTING OF PERISCOPE FOLLOWED BY ATTACK WITH ASROC AND TORPEDOS. SUBMARINE WENT SINKER. LOOSEFOOT 722/723 CONTINUE SEARCH. FOUR BUOY ROAD PLACED BETWEEN CONSTELLATION AND DATUM.

NARR/HAVE CONDUCTED THREE ATTACKS, PROSECUTING WITH A/C

NARR/LOST CONTACT IN PROLONGED FADE

NARR/S3 WAS KILO DELTA AT ONSTA HOWEVER INFLIGHT TROUBLE SHOOTING CORRECTED COMPUTER PROBLEM AT 1019Z. A DISTRIBUTIVE FIELD WAS LAID IN ASSIGNED AREA AND LOFAR CONTACT WAS GAINED ON BY MOST BUOY IN PATTERN. A DIFAR BUOY BEARING INDICATED TGT CUS OF 253T. LSFT 727 WAS VECTORED IN FOR DIP ON CONTACT AT 2035Z, WHICH WAS LATER EVALUATED AS NON SUB. ADDITIONAL LOFAR CONTACT ON NORTHERN BUOYS INDICATED TGT WAS TRACKING 020T WITH A SPEED OF 4.5 KTS CALCULATED FROM MOTORSLOT FREQ. LSFT EXECUTED AIRPLAN 44. HELOS HELD CONTACT FOR A PERIOD OF 15 MINUTES BEFORE GOING COLD. SLVG 702 INVESTIGATED LOFAR PROBABILITY IN SW PORTION OF OPAREA WITH NEGATIVE RESULTS. ENROUTE TO CV SLVG 702 BELIEVED HE HAD A VISUAL SUB BASED S ON WATER DISTURBANCE, WHITE WATER, PLUS SNORKEL EXHAUST STOVE. ACFT DROPPED A SMOKE AND A DIFAR BUT ADP HAD FAILED. UNSUCCESSFULLY ATTEMPTED TO VECTOR HELO TO SINKER POSITION, WHICH WAS 5NM 225T FROM CV. DEPARTED STATION AND RECOVERED CV.

NOMAD

DISCUSSION

NOMAD has been constructed specifically to handle erroneous and malformed input, such as that found in Navy messages. Here is a partial classification of errors treated by NOMAD:

1) Conceptual Problems
   a. Gaps in Meaning
      example: "Midway attacked Kashin." A previous sighting must be inferred.
   b. Goal Conflicts
      example: "Kennedy attacked Nimitz." Unlikely that friends would attack each other.

2) Ambiguity
   a. Multiple Word Meanings
      example: "Returned bombs to Kashin" can mean material transfer or an attack.
   b. Syntactic
      example: "Kashin attacked" can mean that Kashin is the attacker or attackee.
3) Terseness
   a. Missing Actors
      example: "Took under fire with missiles."
   b. Missing Connecting Words
      example: "Intend attack Kashin".
   c. Abbreviations
      example: "est", meaning "establish" or "estimate".
4) Ungrammatical Constructs
   a. Incomplete sentences
      example: "Minor damage to ships at anchor."
   b. Incorrect punctuation
      example: "Agi comm ceased, topside burning, Ramsey joining tf."
   c. Incorrect Verb Use
      example: "open fired".
5) New Words
   (words not in NOMAD's dictionary)
6) Morphology
   a. Spelling Errors
   b. Incorrect Prefix and Suffix Use

The CA, or Conceptual Analyzer [Birnbaum and Selfridge] serves as the foundation of the NOMAD system. Parsing of the input is driven by meaning, rather than by a set of English grammar rules, so that both correct and incorrect input can be handled in the same way. The system's semantic knowledge is stored mainly at the word level, with an MLISP routine corresponding to every word of NOMAD's vocabulary. These routines describe interactions between the word and items that come before or after the word. If the processing for a word cannot be completed when the word is read, its routine creates REQUESTS so that the processing can continue later in the parse. These requests are merely additional routines. When a word has been processed, the system checks to see if requests made by previous words can be satisfied.

For constructing a coherent meaning representation, inference-making routines are called after each word is parsed, to try to fill gaps in the meaning representation. These routines use script-based knowledge that, once again, is embedded in code. A classic example of a script is the ATTACK, in which first a sighting occurs, then projections of weapons or weapon-bearing craft, followed by damages to one side or the other.

NOMAD also implements a scheme for deducing the meaning of unknown words in a sentence, called FOUL-UP [Granger]. Each word-level routine creates expectations for what words can come before it and after it. If a sentence contains unknown words, these expectations are used to fill in the resulting gap in the sentence.

The general algorithm for NOMAD is shown:

```
for each message
    for each sentence
        for each word
            get its dictionary routine.
            invoke the dictionary routine.
            invoke requests from previous words.
            call inferencer to create meaning frames.
            make inferences about gaps in the meaning.
        handle unknown words.
    try alternative interpretations.
```

EXAMPLE

We will outline NOMAD's parse of the message:

LOCKED ON CHALLENGED UNIT OPEN FIRED.

where two sentences are run-together, "challenged" could be a verb or adjective, actors and objects are missing, and verbs are incorrectly conjugated.

To control the parse, NOMAD uses 3 lists: The REQUEST-LIST stores requests made by the word-level routines. The CONCEPT-LIST stores concepts recognized in the input. This is where the meaning representation is built. Concepts are represented by Dependency structures [Schank]. A sample NOMAD Conceptual Dependency structure for the concept of PROJECTION is:

```
PROJECT
        ACTOR  <platform>
        OBJECT <projectile>
        INSTR  <projector>
        TO     <location>
```

(not all fields shown)

where the items in angle brackets are themselves CD structures to be filled in. The ACTOR propels an OBJECT using an INSTR (instrument) TO the destination or target. The TO slot can be filled by a platform. A PROJECT denotes the 'throwing' of an object. In the Navy domain, we use it for a craft firing a projectile. The INFERENCE-LIST builds a meaning representation using both explicit knowledge from the message and inferences derived from it. The three lists start out empty.

Next, the morphological analyzer recognizes the phrase "lock on". The routine LOCK-ON is called, and puts an AIM-CD on the CONCEPT-LIST. An AIM-CD is NOMAD's representation of the act of "aiming" at a target. The routine LOCK-ON also places a request to call LOCK-ON2 later in the parse. The task of LOCK-ON2 is to find the platform being aimed at. After LOCK-ON is done, inferencers determine the context of the current act. The "aim" action is inferred to be part of an "attack", so that an ATTACK FRAME is created. The AIM-CD fills the PREP-FOR-ATTACK slot of the ATTACK FRAME, meaning that the "aim" action is in preparation for an attack. A further inference is that the "attack" is part of a "battle", so that a BATTLE FRAME is also created. At this point in the parse, we have:

```
        REQUEST-LIST =  LOCK-ON2
        CONCEPT-LIST =  AIM-CD
        INFERENCE-LIST =
                BATTLE FRAME
                    EVENTS =
                        ATTACK FRAME
                            PREP-FOR-ATTACK = AIM-CD
```

Next, the word "challenged" is recognized, and the corresponding routine CHALLENGED is invoked. This routine assumes that "challenged" is being used as an adjective in this sentence, and does not contribute to the meaning representation at this point. After CHALLENGED is done, NOMAD tries invoking previous requests. LOCK-ON2 is on the request-list and so it is invoked, but does not succeed in finding an object.

Next, "unit" is parsed. Its default meaning is "ship" and so it invokes the routine SHIP. SHIP causes a SHIP-CD to be added to the concept-list. LOCK-ON2 is invoked again, and this time finds the SHIP-CD on the concept-list and assumes it is the target of the "aim" action. Therefore, the SHIP-CD fills the TO slot of the AIM-CD. The lists now look like:

```
        REQUEST-LIST =  NIL
        CONCEPT-LIST =  AIM-CD
                            TO = SHIP-CD
        INFERENCE-LIST =
                BATTLE FRAME
                    EVENTS =
                        ATTACK FRAME
                            PREP-FOR-ATTACK = AIM-CD
```

Next, the morphological analyzer recognizes the phrase "open fire" in the input. At this point, NOMAD knows that it has a complete concept on the concept-list, and so it assumes that the "firing" action begins a new clause. The routine FIRED is invoked and puts a PROJECT-CD on the concept-list. This is how NOMAD represents firing a projectile at a target. FIRED also places FIRED1 on the request-list. This routine will look for the target of the "firing" action. After FIRED is done, inferencers assume that the "firing" action is a continuation of the attack initiated by the previous "aim" action. The lists look like:

```
        REQUEST-LIST =  FIRED1
        CONCEPT-LIST =  AIM-CD              || PROJECT-CD
```

```
                                TO = SHIP-CD ||
        INFERENCE-LIST =
                BATTLE FRAME
                   EVENTS =
                        ATTACK FRAME
                            PREP-FOR-ATTACK = AIM-CD
                            PROJECT         = PROJECT-CD
```

Finally, the end of a sentence is recognized. FIRED1 is invoked but does not find the target of the "firing" act. At the end of sentence, inference routines traverse the entire meaning representation, trying to fill gaps. Since the actor of the "aim" and "firing" was never specified and since there is no previous context, the message originator (ORIG) is assumed to be the actor. The target of the "firing" is inferred to be the same as the target of the "aim". The final meaning representation on the inference-list looks as follows:

```
        BATTLE FRAME
             PARTICIPANTS =  ORIG, SHIP-CD
             EVENTS       =
                ATTACK FRAME
                    ATTACKER        = ORIG
                    ATTACKEE        = SHIP-CD
                    PREP-FOR-ATTACK = AIM-CD
                                          ACTOR = ORIG
                                          TO    = SHIP-CD
                    PROJECT         = PROJECT-CD
                                          ACTOR = ORIG
                                          TO    = SHIP-CD
```

NOMAD uses the complete meaning representation to generate the following interpretation of the message:

```
                WE AIMED AT AN ENEMY SHIP.
                WE FIRED AT THEM.
```

To further illustrate the NOMAD implementation, we show a sample word-level routine (altered for readability):

```
    PROCEDURE attacked ()
    VARIABLES concept, cd;
    BEGIN

    concept := previous-concept();
    cd      := add-concept(project-cd);

    IF is-platform(concept) THEN
        BEGIN
        message("platform fills the TO slot of PROJECT-CD");
        fill-slot('to, cd, concept);

        message("Waiting for the next concept...");
        expectation('attacked, 'to, 'platform, cd, 'forward);
        request('attacked1, concept, cd);
        END
    ELSE                         % concept was not platform  %
        BEGIN
        message("Waiting for the next concept...");
        expectation('attacked, 'actor, 'platform, cd, 'backward);
        expectation('attacked, 'to,    'platform, cd, 'forward);
        request('attacked2, cd);
        END;
    END;
```

The routine ATTACKED handles the following cases:

```
            PLATFORM ATTACKED PLATFORM.
            PLATFORM ATTACKED.
            ATTACKED PLATFORM.
            ATTACKED.
```

First, it adds a new PROJECT-CD concept to the concept list, since the default meaning for the verb 'attack' is a 'projection', that is, throwing something at a target. Then, ATTACKED checks to see if the previous concept is a platform. If the previous concept is a platform, the routine interprets that as "platform WAS attacked", using the passive form of 'attacked'. The platform is then the destination of the projection, and so it fills the TO slot of the PROJECT-CD. The routine knows, however, that that another platform might follow the word 'attacked', so it creates an expectation that the next concept found might actually be the destination of the PROJECT-CD. (This kind of expectation is only used by the FOUL-UP mechanism when the next word in the input is unknown to NOMAD). ATTACKED then makes a request that the routine ATTACKED1 be used to look for a platform in the input yet to be processed.

If no platform was found before to the word 'attacked', then a request is made to use ATTACKED2 to find a platform in the input yet to be processed. ATTACKED2 thus handles the case "attacked platform".

If the slots of the PROJECT-CD are not filled by the end of the sentence, inferencers will be called to try to fill those slots using information from previous context. In the example above, if ATTACKED1 or ATTACKED2 never found a platform after the word 'attacked', then some of the PROJECT-CD slots would remain unfilled until an inferencer filled them.

NOMAD interacts with the user to make sure that it corrects a sentence as a user intended, and to gain additional information from the user when necessary.

When a meaning representation has been constructed, the generator converts that meaning into a rephrase of the original message. Some sample inputs and outputs for NOMAD are shown (bear in mind that Midway is the vessel sending the message):

SAMPLE RUNS OF NOMAD

```
input:  Periscope sighted by Constellation abt 2000 yds off port
        qtr, and hs helo vectored to datum. Green flares sighted
        after visual on periscope.  Hs gained active ctc and
        conducted two attacks.
output
first:  The Constellation identified an enemy submarine that was
        at 225 degrees 2000 yards from its location.  A helicopter
        squadron pursued the enemy submarine.  The helicopter
        squadron identified some green flares.  By using an active
        sonar, the helicopter squadron identified the enemy
        submarine, and they fired twice at the enemy submarine.
best:   (same as above, except 3rd line, which reads:
        The Constellation identified some green flares.)

input:  Took under fire with missiles.
output: We launched some missiles.

input:  Two destroyers were sighted.
output: We sighted 2 enemy destroyers.

input:  Midway fired talos at fanning.
output: We launched a talos missile at the Fanning
        [ a fast-frigate1076].

input:  Returned bombs to kashin.
output
first:  We took some bombs to the location of a Kashin destroyer.
best:   We dropped some bombs on a Kashin destroyer.

input:  Midway sighted kashin. fired.
output: We sighted a Kashin destroyer. We fired at them.

input:  Visual detection of enemy aircraft.
output: By using our eye, we sighted an enemy aircraft.

input:  Midway gained radar contact on unknown ship.
output: By using our radar, we gained contact with an enemy ship.

input:  Midway contacted submarine by radio.
output: We communicated with an enemy submarine by a radio.

input:  We did not locate enemy ship.
output: We did not sight an enemy ship.

input:  Midway engaged an enemy ship with guns.
output: We fired our guns at an enemy ship.

input:  Midway unsuccessfully attacked enemy ship.
output: We fired at an enemy ship, but missed.

input:  Midway fired at enemy ship and missed.
output: We fired at an enemy ship, but missed.
```

```
input:  Pegasus sighted in port.
output: We sighted a Pegasus patrol-hydrofoil.

input:  Midway destroyed enemy ship with missiles.
output: We launched some missiles at an enemy ship. We noticed
        that they were damaged beyond repair.

input:  Midway attacked enemy with missiles. attack failed.
output: We launched some missiles at an enemy ship, but missed..

input:  Midway responded with missile fire.
output: We retaliated and launched a missile.

input:  Lamps vectored to investigate.
output: A lamps helicopter squadron left their present location
        so that they could patrol.

input:  Assumed position onsta.
output: We went to designated-station.

input:  Kashin continued firing at pegasus.
output: A Kashin destroyer continued firing at a Pegasus
        patrol hydrofoil.

input:  Kobchik will fire at wainwright.
output: The Kobchik [ a Riga frigate ] will fire at the
        Wainright [ a cruiser ].
```

CRITIQUE OF NOMAD

NOMAD has proved to be a capable analyzer of malformed input. Word-level processing is a powerful tool for understanding English, since it permits flexible handling of ungrammatical and ambiguous constructs.

Lack of extensibility is the most important drawback of NOMAD. Since the system's knowledge is mainly imbedded in word-level routines, adding a new word to the system requires writing a new routine. The new routine should ideally take into account interactions with all the word-level routines already present in the system. Further, the routines already present in the system may need to be modified so as to interact well with the new routine.

In practice, we do not check every routine when a new word is added. Rather, we test the system and make corrections only when a bad interaction is found. Thus, the system is not guaranteed to be self-consistent. Since NOMAD has several hundred words in its vocabulary, it is impractical to check the entire system when a new word is added.

Encoding grammatic knowledge at the word-level is also cumbersome. For example, the routine for nearly every verb makes its own checks for active or passive usage. A more centralized grammatic mechanism would eliminate this kind of redundancy.

In principle, the knowledge currently encoded in the word-level routines could be made declarative (that is, stored as data), so as to be usable by other parts of the system, especially by the English generation module.

Another concern with NOMAD is that it cannot distinguish correct from incorrect input. NOMAD merely tries to come up with the best meaning it can in every case. A more intelligent system would be sensitive to when the parse appears to be "going wrong" and could use such knowledge to reparse the message differently or to indicate that an error occurred.

With word-level routines, there are also problems of how far backward or forward a concept should be searched for, relative to the current word. If one searches too broadly, one may find a desired concept outside the current context; if too narrowly, the concept may be present and not reached. For example, in the "attacked" example described above, the requested routines attacked1() and attacked2() will be called until another platform is detected in the sentence. It might well be that the new platform is in an unrelated phrase to the one in which the verb "attacked" appeared.

VOX

To make the NOMAD system more extensible, we are currently building a version that uses, not word-level analysis, but PHRASAL ANALYSIS. We call the new system VOX, which stands for VOcabulary eXtension system. Our goal is to make this system extensible by interaction with a user, rather than by adding to the database programmatically. Our ideas about phrasal analysis originate from the work on PHRAN [Wilensky and Arens]. Phrasal analysis consists in matching the input to one or more phrase-level patterns stored in a knowledge database. When the input has been matched, it is said to be understood. Semantic actions can be associated with each phrase, so that whenever a phrase is matched to part of the input, a corresponding meaning representation for the phrase may be constructed.

To extend the knowledge base of the system, we simply add new patterns to the database. Ideally, patterns are independent entities whose interaction introduces no side-effects, so that new phrases can be easily added to or removed from the database.

We illustrate our phrasal analysis scheme using the same example as for NOMAD:

LOCKED ON CHALLENGED UNIT OPEN FIRED.

Assume that the database contains the following phrasal patterns, among others:

<u>PATTERN</u>		<u>REDUCES TO</u>
LOCK	->	ACT
ON	->	STATE
ON	->	PREP
OPEN	->	ACT
FIRE	->	ACT
FIRE	->	NOUN
LOCK ON	->	AIM-AT
AIM-AT	->	CONCEPT
OPEN FIRE	->	PROJECT
OPEN FIRE	->	CONCEPT
PROJECT	->	CONCEPT
CHALLENGE	->	ADJ
CHALLENGE	->	VERB
UNIT	->	PLATFORM
PLATFORM	->	NOUN
ADJ NOUN	->	[pattern under NOUN]
ADJ NOUN	->	NP
AIM-AT PLATFORM	->	CONCEPT
PLATFORM ACT	->	CONCEPT
PLATFORM ACT PLATORM	->	CONCEPT

When a pattern 'reduces' to another pattern, it SUGGESTS the second pattern. Not all the patterns shown nor all their 'reductions' will be used in the parse, though they may be considered by the phrasal analysis algorithm. We will show only patterns that have been chosen as the meaning of the input at every point in the parse. The parse proceeds as follows:

After morphological analysis of "locked", the pattern LOCK is found by lookup. Several such patterns might be found, each representing a different meaning of the verb "to lock". The algorithm must choose a particular one as representing the meaning-so-far. Say it chooses the one corresponding to locking a door:

```
((EVENT LOCK)
 (TENSE PAST))
```

Next, the pattern ON is found by lookup. The analyzer notices that the entire input can be covered by extending the pattern LOCK from the previous word, to give the pattern LOCK ON. This pattern has the associated meaning

```
((EVENT AIM)
 (TENSE PAST))
```

The past tense was obtained by looking for relevant properties in the pattern LOCK which was extended. Note that the meaning "to lock a door" is discarded. LOCK ON further reduces to the pattern AIM-AT, which represents a general "aiming" action. Both these patterns cover the input up to this point.

Next, the pattern CHALLENGE is found by lookup. Its associated meaning is:

```
((EVENT CHALLENGE)
 (TENSE PAST))
```

CHALLENGE reduces to ADJ, a general adjective, and this gets the meaning:

```
(DESCRIPTION
 ((EVENT CHALLENGE)
  (TENSE PAST)))
```

Neither CHALLENGE nor ADJ can extend patterns from the previous word in the input.

Next, the pattern UNIT is found by lookup. Its default meaning in the Navy domain is a ship. The phrasal parser notices that it can cover the entire input with a single pattern as follows. Reduce UNIT to ship, giving the meaning

```
((ACTOR PLATFORM)
 (KIND SHIP))
```

Reduce PLATFORM to NOUN, copying the meaning for PLATFORM. Extend the previous pattern ADJ to give ADJ NOUN, copying the meanings of ADJ and NOUN. Reduce ADJ NOUN to the pattern which gave rise to NOUN, that is, to UNIT. This pattern UNIT covers the input "challenged unit", and has the following meaning built for it:

```
((ACTOR SHIP)
 (DESCRIPTION
    ((EVENT CHALLENGE)
     (TENSE PAST))))
```

Reduce UNIT to PLATFORM, building the meaning:

```
((ACTOR PLATFORM)
 (KIND SHIP)
 (DESCRIPTION
    ((EVENT CHALLENGE)
     (TENSE PAST))))
```

PLATFORM extends the pattern AIM-AT to give AIM-AT PLATFORM, with meaning

```
((EVENT AIM)
 (TENSE PAST)
 (OBJECT
      ((ACTOR PLATFORM)
       (KIND SHIP)
       (DESCRIPTION
          ((EVENT CHALLENGE)
           (TENSE PAST)))) ))
```

AIM-AT PLATFORM reduces to CONCEPT. At this point, the phrasal analyzer realizes a complete concept has been recognized. Inferencers are invoked to check for gaps in the meaning representation. They find no actor for the "aim" event, and so add a default actor ORIG, the message originator.

In a similar way to "locked on", "open fired" is recognized as the pattern OPEN FIRE, with its associated meaning:

```
((EVENT PROJECT)
 (TENSE PAST))
```

The phrasal analyzer again finds a pattern that covers the entire input: OPEN FIRE reduces to CONCEPT, which extends the previous CONCEPT to give CONCEPT-LIST. The meanings of the two CONCEPTs are merged and inference mechanisms fill in missing information, to give the final representation:

```
                [((EVENT AIM)
                  (TENSE PAST)
                  (ACTOR ORIG)
                  (OBJECT  A))

                 ((EVENT PROJECT)
                  (TENSE PAST)
                  (ACTOR ORIG)
                  (OBJECT A))]


        where   A =   ((ACTOR PLATFORM)
                       (KIND SHIP)
                       (DESCRIPTION
                          ((EVENT CHALLENGE)
                           (TENSE PAST))))
```

The final meaning representation can be used by a generator to generate much the same rephrase of the message as was shown for the NOMAD generator.

We summarize the patterns chosen during the parse:

CRITIQUE OF VOX

Work on the VOX system has begun recently, so not all of the program modules have been constructed as yet. A phrasal analyzer is up and working, and a rudimentary meaning construction apparatus has also been implemented. Yet even in our primitive system, adding new patterns by hand has proved simple, and has produced no side effects.

We expect the VOX system to be more extensible than systems that have no declarative phrasal knowledge. Even if extension of the database by interaction with a user proves hard to attain, we feel that programmatic extension of the system will not be hard for a large part of the Naval domain.

In contrast to NOMAD, VOX will have much more declarative knowledge. As Wilensky has suggested, we plan to use the same knowledge for both analysis and generation of text.

Another difference between NOMAD and VOX is that NOMAD usually doesn't know if an input is malformed. Any two words in English can be adjacent in a sentence, as far as NOMAD is concerned. VOX, on the other hand, can tell when an input cannot match any of its patterns. If that is the case, then either the input is malformed or VOX does not have the knowledge to handle it. In either case, VOX knows that something has gone wrong, whereas NOMAD does not.

With VOX, there is an unfortunate tendency to resolve English problems merely by adding new patterns. This can quickly get out of hand, however. We do not wish to add all English sentences of ten words or less into the database, for example. We must strive to find the right levels of generality for patterns that we add to the database. Certainly there are many literal patterns, but more often a general pattern can be used instead of a more specific one. Generality is important in handling novel input. We should add patterns only when there is a positive reason for doing so, when a nuance in the meaning of a phrase cannot be deduced from the components of the phrase.

We further see no reason why phrasal techniques cannot be successfully applied to any domain for which an English understanding system is desired. If we can build an interactively extensible VOX, then we will have taken steps toward a system which can learn language.

CONCLUSION

NOMAD was a first cut. VOX is a more sophisticated attempt at understanding English. These systems indicate that both word-level and phrase-level knowledge are essential to understanding natural language.

We have found also that much can be learned about processing correct input by processing erroneous input. The line between correct and incorrect English is often unclear, so that a system which cannot handle erroneous input is of limited use.

REFERENCES

Bob Bechtel
Elements of Naval Domain Knowledge.
Bob Bechtel
NOSC Working Paper
April 2, 1981

L. Birnbaum and M. Selfridge
Problems in the Conceptual Analysis of Natural Language.
Research Report #168. 1979.
Dept. of Computer Science,
Yale University
New Haven, Connecticut.

R. A. Dillard
Text-Understanding Techniques Applied to Partly Formatted Navy Tactical Messages.
NOSC Technical Document 405
San Diego, California.
December 17, 1980.

Richard H. Granger
FOUL-UP: A Program That Figures Out Meanings of Words From Context.
IJCAI, August 22, 1977, vol. 1.
MIT,
Cambridge, MA.

David M. Keirsey
Natural Language Processing Applied to Navy Tactical Messages.
NOSC Technical Document 324
San Diego, California.
February 1, 1980.

J. Moore, ed.
Jane's Fighting Ships, 1976-1977.
Franklin Watts, Inc., 1976.

John V. Noel, Jr. and Edward L. Beach,
Naval Terms Dictionary.
United States Naval Institute,
Annapolis, Maryland. 1971, 3rd ed.

Michael J. Pazzani,
Word Sense Disambiguation.
Computer Science Technical Report, TR-CS-12-80.

MS Thesis.
University of Connecticut, 1980.

Roger Schank and R. Abelson,
Scripts, Plans, Goals, and Understanding.
Lawrence Erlbaum Associates, 1977.

Bill Wedertz, ed.
DICNAVAB: Dictionary of Naval Abbreviations.
1970 Edition
The United States Naval Institute,
Annapolis, Maryland.

Robert Wilensky and Yigal Arens
PHRAN: A Knowledge-based Approach to Natural Language Analysis.
UCB/ERL M80/34
August 12, 1980
Electronics Research Laboratory
University of California,
Berkeley, California 94720