Generation that Exploits Corpus-Based Statistical Knowledge

Irene Langkilde and **Kevin Knight**
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
ilangkil@isi.edu and knight@isi.edu

Abstract

We describe novel aspects of a new natural language generator called Nitrogen. This generator has a highly flexible input representation that allows a spectrum of input from syntactic to semantic depth, and shifts the burden of many linguistic decisions to the statistical post-processor. The generation algorithm is compositional, making it efficient, yet it also handles non-compositional aspects of language. Nitrogen's design makes it robust and scalable, operating with lexicons and knowledge bases of one hundred thousand entities.

1 Introduction

Language generation is an important subtask of applications like machine translation, human-computer dialogue, explanation, and summarization. The recurring need for generation suggests the usefulness of a general-purpose, domain-independent natural language generator (NLG). However, "plug-in" generators available today, such as FUF/SURGE (Elhadad and Robin, 1998), MUMBLE (Meteer et al., 1987), KPML (Bateman, 1996), and CoGenTex's RealPro (Lavoie and Rambow, 1997), require inputs with a daunting amount of linguistic detail. As a result, many client applications resort instead to simpler template-based methods.

An important advantage of templates is that they sidestep linguistic decision-making, and avoid the need for large complex knowledge resources and processing. For example, the following structure could be a typical result from a database query on the type of food a venue serves:

```
((:obj-type venue)(:obj-name Top_of_the_Mark)
 (:attribute food-type)(:attrib-value American))
```

By using a template like

```
<obj-name> 's <attribute> is <attrib-value>.
```

the structure could produce the sentence, "Top of the Mark's food type is American."

Templates avoid the need for detailed linguistic information about lexical items, part-of-speech tags, number, gender, definiteness, tense, sentence organization, sub-categorization structure, semantic relations, etc., that more general NLG methods need to have specified in the input (or supply defaults for). Such information is usually not readily inferrable from an application's database, nor is it always readily available from other sources, with the breadth of coverage or level of detail that is needed. Thus, using a general-purpose generator can be formidable (Reiter, 1995). However, templates only work in very controlled or limited situations. They cannot provide the expressiveness, flexibility or scalability that many real domains need.

A desirable solution is a generator that abstracts away from templates enough to provide the needed flexibility and scalability, and yet still requires only minimal semantic input (and maintains reasonable efficiency). This generator would take on the responsibility of finding an appropriate linguistic realization for an underspecified semantic input. This solution is especially important in the context of machine translation, where the surface syntactic organization of the source text is usually different from that of the target language, and the deep semantics are often difficult to obtain or represent completely as well. In Japanese to English translation, for example, it is often hard to determine from a Japanese text the number or gender of a noun phrase, the English equivalent of a verb tense, or the deep semantic meaning of sentential arguments. There are many other obvious syntactic divergences as well.

Thus, shifting such linguistic decisions to the generator is significantly helpful for client applications. However, at the same time, it imposes enormous needs for knowledge on the generator program. Traditional large-scale NLG already requires immense amounts of knowledge, as does any large-scale AI enterprise. NLG operating on a scale of 200,000 entities (concepts, relations, and words) requires large and sophisticated lexicons, grammars, ontologies, collocation lists, and morphological tables. Acquiring and applying accurate, detailed knowledge of this breadth poses difficult problems.

(Knight and Hatzivassiloglou, 1995) suggested

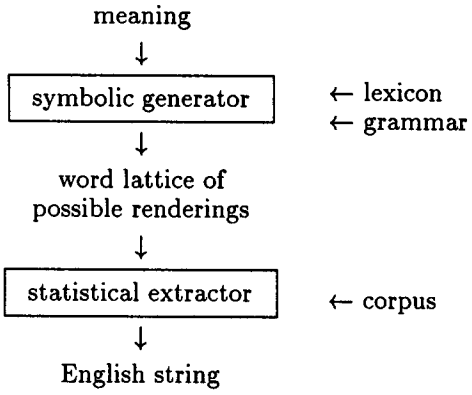

Figure 1: Combining Symbolic and Statistical Knowledge in a Natural Language Generator (Knight and Hatzivassiloglou, 1995).

overcoming this knowledge acquisition bottleneck in NLG by tapping the vast knowledge inherent in English text corpora. Experiments showed that corpus-based knowledge greatly reduced the need for deep, hand-crafted knowledge. This knowledge, in the form of n-gram (word-pair) frequencies, could be applied to a set of semantically related sentences to help sort good ones from bad ones. A corpus-based statistical ranker takes a set of sentences packed efficiently into a *word lattice*, (a state transition diagram with links labeled by English words), and extracts the best path from the lattice as output, preferring fluent sentences over contorted ones. A generator can take advantage of this by producing a lattice that encodes various alternative possibilities when the information needed to make a linguistic decision is not available.

Such a system organization shown in Figure 1, is robust against underspecified and even ambiguous input meaning structures. Traditionally, underspecification is handled with rigid defaults (e.g., assume present tense, use the alphabetically-first synonyms, use nominal arguments, etc.). However, the word lattice structure permits all the different possibilities to be encoded as different phrasings, and the corpus-based statistical extractor can select a good sentence from these possibilities.

The questions that still remain are: What kind of input representation is minimally necessary? What kinds of linguistic decisions can the statistics reliably make, and which instead need to be made symbolically? How should symbolic knowledge be applied to the input to efficiently produce word lattices from the input?

This paper describes Nitrogen, a generation system that computes word lattices from a meaning representation to take advantage of corpus-based statistical knowledge. Nitrogen performs sentence realization and some components of sentence planning—namely, mapping domain concepts to content words, and to some extent, mapping semantic relations to grammatical ones. It contributes:

- A flexible input representation based on conceptual meanings and the relations between them.
- A new grammar formalism for defining the mapping of meanings onto word lattices.
- A new efficient algorithm to do this mapping.
- A large grammar, lexicon, and morphology of English, addressing linguistic phenomena such as knowledge acquisition bottlenecks and underspecified/ambiguous input.

This paper is organized as follows. First, we describe our Abstract Meaning Representation language (AMR). Then we outline the generation algorithm and describe how various knowledge sources apply to render an AMR into English, including lexical, morphological, and grammatical knowledge bases. We describe the structure of these knowledge bases and give examples. We also present a technique that adds powerful flexibility to the grammar formalism. We finish with a discussion of the strengths and weaknesses of our generation system.

2 Abstract Meaning Representation

The AMR language is composed of concepts from the SENSUS knowledge base (Knight and Luk, 1994), including all of WordNet 1.5 (Miller, 1990), and keywords relating these concepts to each other.[1]

An AMR is a labeled directed graph, or feature structure, derived from the PENMAN Sentence Plan Language (Penman, 1989). The most basic AMR is of the form `(label / concept)`, e.g.:[2]

```
(m1 / |dog<canid|)
```

The slash is shorthand for a type (or instance) feature, and in logic notation this AMR might be written as *instance(m1, dog)*. This AMR can represent "the dog," "the dogs," "a dog," or "dog," etc.

A concept can be modified using keywords:

```
(m2 / |dog<canid|
   :quant plural)
```

[1]Strings can be used in place of concepts. If the string is not a recognized word/phrase, then the generator will add this ambiguity to the word lattice for the statistical extractor to resolve by proposing all possible part-of-speech tags. We prefer to use concepts because they make the AMR more language-independent, and enable semantic reasoning and inference.

[2]Concept names appear between vertical bars. We use a set of short, unique concept names derived from the structure of WordNet by Jonathan Graehl, and available from http://www.isi.edu/natural-language/GAZELLE.html

This narrows the meaning to "the dogs," or "dogs."

Concepts can be associated with each other in a nested fashion to form more complex meanings. These relations between conceptual meanings are also expressed through keywords. It is through them that our formalism exhibits an appealing flexibility. A client has the freedom to express the relations at various semantic and syntactic levels, using whichever level of representation is most convenient.[3] We have currently implemented shallow semantic versions of roles such as :agent, :patient, :sayer, :sensor, etc., as well as deep syntactic roles such as :oblique1, :oblique2, and :oblique3 (which correspond to deep subject, object, and indirect object respectively, and serve as an abstraction for passive versus active voice) and the straightforward syntactic roles :subject, :direct-object, :indirect-object, etc. We explain further how this is implemented later in the paper.

Below is an example of a slightly more complex meaning. The root concept is eating, and it has an agent and a patient, which are dogs and a bone (or bones), respectively.

```
(m3 / |eat,take in|
   :agent (m4 / |dog<canid|
              :quant plural)
   :patient (m5 / |os,bone|))
```

Possible output includes "The dogs ate the bone," "Dogs will eat a bone," "The dogs eat bones," "Dogs eat bone," and "The bones were eaten by dogs."

3 Lexical Knowledge

The Sensus concept ontology is mapped to an English lexicon that is consulted to find words for expressing the concepts in an AMR. The lexicon is a list of 110,000 tuples of the form:

```
(<word> <part-of-speech> <rank> <concept>)
Examples:
(("eat" VERB 1 |eat,take in|)
 ("eat" VERB 2 |eat>eat lunch|)
  ... )
```

The <rank> field orders the concepts by sense frequency for the given word, with a lower number signifying a more frequent sense.

Like other types of knowledge used in Nitrogen, the lexicon is very simple. It contains no information about features like transitivity, subcategorization, gradability (for adjectives), or countability (for nouns), etc. Such features are needed in other generators to produce correct grammatical constructions. Our statistical post-processor instead more softly (and robustly) ranks different grammatical realizations according to their likelihood.

At the lexical level, several important issues in word choice arise. WordNet maps a concept to one or more synonyms. However, some words may be less appropriate than others, or may actually be misleading in certain contexts. An example is the concept |sell<cozen| to which the lexicon maps the words "betray" and "sell." However, it is not very common to use the word "sell" in the sense of "A traitor sells out on his friends." In the sentence "I cannot |sell<cozen| their trust" the word "sell" is misleading, or at least sounds very strange; "betray" is more appropriate.

This word choice problem occurs frequently, and we deal with it by taking advantage of the word-sense rankings that the lexicon offers. According to the lexicon, the concept |sell<cozen| expresses the second most frequent sense of the word "betray," but only the sixth most frequent sense of the word "sell." To minimize the lexical choice problem, we have adopted a policy of rejecting words whose primary sense is not the given concept when better words are available.[4]

Another issue in word choice relates to the broader issue of preserving ambiguities in MT. In source language analysis, it is often difficult to determine which concept is intended by a certain word. The AMR allows several concepts to be listed together in a disjunction. For example,

```
(m6 / (*OR* |sell<cozen| |cheat on| |bewray|
             |betray,fail| |rat on|))
```

The lexical lookup will attempt to preserve the ambiguity of this *OR*. If it happens that several or all of the concepts in a disjunction can be expressed using the same word, then the lookup will return only that word or words in preference to the other possibilities. For the example above, the lookup returns only the word "betray." This also reduces the complexity of the final sentence lattices.

4 Morphological Knowledge

The lexicon contains words in their root form, so morphological inflections must be generated. The system also performs derivational morphology, such as adjective→noun and noun→verb (ex:

[3] This flexibility has another advantage from a research point of view. We consider the appropriate level of abstraction an important problem in interlingua-style machine translation. The flexibility of this representation allows us to experiment with various levels of abstraction without changing the underlying system. Further, it has opened up to us the possibility of implementing interlingua-based semantic transfer, where the interlingua serves as the transfer mechanism, rather than being a single, fixed peak point of abstraction.

[4] A better "soft" technique would be to accept all words returned by the lexicon for a given concept, but associate with each word a preference score using a method such as Bayes' Rule and probabilities computed from a corpus such as SEMCOR, allowing the statistical extractor to choose the best alternative. We plan to implement this in the future.

"translation"→"translate") to give the generator more syntactic flexibility in expressing complex AMR's. This flexibility ensures that the generator can find a way to express a complex meaning represented by nested AMRs, but is also useful for solving problems of syntactic divergence in MT.

Both kinds of morphology are handled the same way. Rules and exception tables are merged into a single, concise knowledge base. Here, for example, is a portion of the table for pluralizing nouns:

```
("-child" "children")
("-person" "people" "persons")
("-a" "as" "ae")        ; formulas/formulae
("-x" "xes" "xen")      ; boxes/oxen
("-man" "mans" "men") ; humans/footmen
("-Co" "os" "oes")
```

The last line means: if a noun ends in a consonant followed by "-o," then we compute two plural forms, one ending in "-os" and one ending in "-oes," and put both possibilities in the word lattice for the post-generation statistical extractor to choose between later. Deciding between these usually requires a large word list. However, the statistical extractor already has a strong preference for "photos" and "potatoes" over "photoes" and "potatos," so we do not need to create such a list. Here again corpus-based statistical knowledge greatly simplifies the task of symbolic generation.

Derivational morphology raises the issue of meaning shift between different part-of-speech forms (such as "depart"→ "departure"/"department"). Errors of this kind are infrequent, and are corrected in the morphology tables.

5 Generation Algorithm

An AMR is transformed into word lattices by keyword-based grammar rules described in Section 7. By contrast, other generators organize their grammar rules around syntactic categories. A keyword-based organization helps achieve simplicity in the input specification, since syntactic information is not required from a client. This simplification can make Nitrogen more readily usable by client applications that are not inherently linguistically oriented. The decisions about how to syntactically realize a given meaning can be left largely up to the generator.

The top-level keywords of an AMR are used to match it with a rule (or rules). The algorithm is compositional, avoiding a combinatorial explosion in the number of rules needed for the various keyword combinations. A matching rule splits the AMR apart, associating a sub-AMR with each keyword, and lumping the relations left over into a sub-AMR under the :rest role using the same root as the original AMR. Each sub-AMR is itself recursively matched against the keyword rules, until the recursion bottoms out at a basic AMR which matches with the *instance* rule.

Lexical and morphological knowledge is used to build the initial word lattices associated with a concept when the recursion bottoms out. Then the instance rule builds basic noun and verb groups from these, as well as basic word lattices for other syntactic categories. As the algorithm climbs out of the recursion, each rule concatenates together lattices for each of the sub-AMR's to form longer phrases. The rhs specifies the needed syntactic category for each sub-lattice and the surface order of the concatenation, as well as the syntactic category for the new resulting lattice. Concatenation is performed by attaching the end state of one sub-lattice to the start state of the next. Upon emerging from the top-level rule, the lattice with the desired syntactic category, by default S (sentence), is selected and handed to the statistical extractor for ranking.

The next sections describe further how lexical and morphological knowledge are used to build the initial word lattices, how underspecification is handled, and how the grammar is encoded.

6 The Instance Rule

The instance rule is the most basic rule since it is applied to every concept in the AMR. This rule builds the initial word lattices for each lexical item and for basic noun and verb groups. Each concept in the AMR is eventually handed to the instance rule, where word lattices are constructed for all available parts of speech.

The relational keywords that apply at the instance level are :polarity, :quant, :tense, and :modal. In cases where a meaning is underspecified and does not include these keywords, the instance rule uses a *recasting* mechanism (described below) to add some of them. If not specified, the system assumes positive polarity, both singular and plural quantities, all possible time frames, and no modality.

Japanese nouns are often ambiguous with respect to number, so generating both singular and plural possibilities and allowing the statistical extractor to choose the best one results in better translation quality than rigidly choosing a single default as traditional generation systems do. Allowing number to be unspecified in the input is also useful for general English generation as well. There are many instances when the number of a noun is dictated more by usage convention or grammatical constraint than by semantic content. For example, "The company has (a plan/plans) to establish itself in February," or "This child won't eat any carrots," ("carrots" must be plural by grammatical constraint). It is easier for a client program if the input is not required to specify number in these cases, but is allowed to rely

on the statistical extractor to supply the best one.

In translation, there is frequently no direct correspondence between tenses of different languages, so in Nitrogen, tense can be coarsely specified as either past, present, or future, but need not be specified at all. If not specified, Nitrogen generates lattices for the most common English tenses, and allows the statistical extractor to choose the most likely one.

The instance rule is factored into several subinstance rules with three main categories: nouns, verbs, and miscellaneous. The noun instance rules are further subdivided into two rules, one for plural noun phrases, and the other for singular. The verb instance rules are factored into two categories relating to modality and tense.

Polarity can apply across all three main instance categories (noun, verb, and other), but only affects the level it appears in. When applied to nouns or adjectives, the result is "non-" prepended to the word, which conveys the general intention, but is not usually very grammatical. Negative polarity is usually most fluently expressed in the verb rules with the word "not," e.g., "does not eat."[5]

7 Grammar Formalism

The grammatical specifications in the keyword rules constitute the main formalism of the generation system. The rules map semantic and syntactic roles to grammatical word lattices. These roles include:

```
:agent, :patient, :domain, :range, :source,
:destination, :spatial-locating,
:temporal-locating, :accompanier;
:oblique1, :oblique2, :oblique3;
:subject, :object, :mod, etc.
```

A simplified version of the rule that applies to an AMR with `:agent` and `:patient` roles is:

```
((x1 :agent)
 (x2 :patient)
 (x3 :rest)
 ->
 (s  (seq (x1 np nom-pro) (x3 v-tensed)
          (x2 np acc-pro)))
 (s  (seq (x2 np nom-pro) (x3 v-passive)
          (wrd "by") (x1 np acc-pro)))
 (np (seq (x3 np acc-pro nom-pro) (wrd "of")
      (x2 np acc-pro) (wrd "by") (x1 np acc-pro)))
 (s-ger (seq ...))
 (inf (seq ...)))
```

The left-hand side is used to match an AMR with agent and patient roles at the top level. The `:rest` keyword serves as a catch-all for other roles that appear at the top level. Note that the rule specifies two ways to build a sentence, one an active voice version and the other passive. Since at this level the input may be underspecified regarding which voice to use, the statistical extractor is expected to choose later the most fluent version. Note also that this rule builds lattices for other parts of speech, in addition to sentences (ex: "the consumption of the bone by the dogs"). In this way the generation algorithm works bottom-up, building lattices for the leaves (innermost nested levels of the input) first, to be combined at outer levels according the relations between the leaves. For example, the AMR below will match this rule:

```
(m7 / |eat,take in|
    :time present
    :agent (d / |dog,canid|
              :quant plural)
    :patient (b / |os,bone|
                :quant sing))
```

Below are some sample lattices that result from applying the rule above to this AMR:[6]

```
(S (or (seq (or (wrd "the") (wrd "*empty*"))
            (wrd "dog") (wrd "+plural")
            (wrd "may") (wrd "eat")
            (or (wrd "the") (wrd "a")
                (wrd "an") (wrd "*empty*"))
            (wrd "bone"))
       (seq (or (wrd "the") (wrd "a")
                (wrd "an") (wrd "*empty*"))
            (wrd "bone") (wrd "may") (wrd "be")
            (or (wrd "being") (wrd "*empty*"))
            (wrd "eat") (wrd "+pastp") (wrd "by")
            (or (wrd "the") (wrd "*empty*"))
            (wrd "dog") (wrd "+plural"))))
(NP (seq (or (wrd "the") (wrd "a")
             (wrd "an") (wrd "*empty*"))
         (wrd "possibility") (wrd "of")
         (or (wrd "the") (wrd "a")
             (wrd "an") (wrd "*empty*"))
         (wrd "consumption") (wrd "of")
         (or (wrd "the") (wrd "a")
             (wrd "an") (wrd "*empty*"))
         (wrd "bone") (wrd "by")
         (or (wrd "the") (wrd "*empty*"))
         (wrd "dog") (wrd "+plural"))))
(S-GER ...)
(INF ...)
```

Note the variety of symbolic output that is produced with these excessively simple rules. Each relation is mapped not to one but to many different realizations, covering regular and irregular behavior exhibited in natural language. Purposeful overgeneration becomes a strength.

[5] We plan to generate more fluent expressions for negative polarity on nouns and adjectives, for example, "unhappy" instead of "non-happy."

[6] The grammar rules can insert the special token `*empty*`, here indicating an option for the null determiner. Before running, the statistical extractor removes all `*empty*` transitions by determinizing the word lattice. Note also the insertion of morphological tokens like `+plural`. Inflectional morphology rules also apply during this determinizing stage.

The `:rest` keyword in the rule head provides a handy mechanism for decoupling the possible keyword combinations. By means of this mechanism, keywords which generate relatively independent word lattices can be organized into separate rules, avoiding combinatorial explosion in the number of rules which need to be written.

7.1 Recasting Mechanism

The *recasting* mechanism that is used in the grammar formalism gives it unique power and flexibility. The recasting mechanism enables the generator to transform one semantic representation into another one (such as deep to shallow, or instance to sub-instance) and to accept as input a specification anywhere along this spectrum, permitting meaning to be encoded at whatever level is most convenient. The recasting mechanism also makes it possible to handle non-compositional aspects of language.

One area in which we use this mechanism is in the `:domain` rule. Take for example the sentence, "It is necessary that the dog eat." It is sometimes most convenient to represent this as:

```
(m8 / |obligatory<necessary|
   :domain (m9 / |eat,take in|
              :agent (m10 / |dog,canid|)))
```

and at other times as:

```
(m11 / |have the quality of being|
   :domain (m12 / |eat,take in|
              :agent (d / |dog,canid|))
   :range (m13 / |obligatory<necessary|))
```

but we can define them to be semantically equivalent. In our system, both are accepted, and the first is automatically transformed into the second.

Other ways to say this sentence include "The dog is required to eat," or "The dog must eat." However, the grammar formalism cannot express this, because it would require inserting the word lattice for `|obligatory<necessary|` *within* the lattice for m9 or m12—but the formalism can only concatenate lattices. The recasting mechanism solves this problem, by recasting the above AMR as:

```
(m14 / |eat,take in|
   :modal (m15 / |obligatory<necessary|)
   :agent (m16 / |dog,canid|))
```

which makes it possible to form these sentences. The syntax for recasting the first AMR to the second is:

```
((x1 :rest)
 (x2 :domain)
 ->
 (? (x1 (:new (/ |have the quality of being|)
    (:domain x2) (:range x1)) ?))
```

and for recasting the second into the third:

```
((x1 :rest)
 (x2 :domain)
 (x3 :range)
 ->
 (? (x2 (:add (:modal (x3 (:add (:extra x1))))) ?))
 (s (seq (x2 np nom-pro) (x1 v-tensed)
     (x3 adj np acc-pro)))
 (s (seq (wrd "it") (x1 v-tensed)
     (x3 adj np acc-pro) (wrd "that") (x2 s)))
 ...)
```

The `:new` and `:add` keywords signal an AMR recast. The list after the keyword contains the instructions for doing the recast. In the first case, the `:new` keyword means: build an AMR with a new root, `|have the quality of being|`, and two roles, one labeled `:domain` and assigned sub-AMR `x2`; the other labeled `:range` and assigned sub-AMR `x1`. The question mark causes a direct splice of the results from the recast.

In the second case, the `:add` keyword means: insert into the sub-AMR of `x2` a role labeled `:modal` and assign to it the sub-AMR of `x3` which itself is recast to include the roles in the sub-AMR of `x1` but not its root. (This is in case there are other roles such as polarity or time which need to be included in the new AMR.)

In fact, recasting makes it possible to nest modals within modals to any desired depth, and even to attach polarity and tense at any level. For example, "It is not possible that it is required that you are permitted to go," can be also (more concisely) stated as "It cannot be required that you be permitted to go," or "It is not possible that you must be permitted to go," or "You cannot have to be permitted to go." This is done by a grammar rule expressing the most nested modal concept as a modal verb and the remaining modal concepts as a combination of regular verbs or adjective phrases. Our grammar includes a fairly complete model of obligation, possibility, permission, negation, tense, and all of their possible interactions.

8 Discussion

We have presented a new generation grammar formalism capable of mapping meanings onto word lattices. It includes novel mechanisms for constructing and combining word lattices, and for re-writing meaning representations to handle a broad range of linguistic phenomena. The grammar accepts inputs along a continuum of semantic depth, requiring only a minimal amount of syntactic detail, making it attractive for a variety of purposes.

Nitrogen's grammar is organized around semantic input patterns rather than the syntax of English. This distinguishes it from both unification grammar (Elhadad, 1993a; Shieber et al., 1989) and systemic-network grammar (Penman, 1989). Meanings can

be expressed directly, or else be recast and recycled back through the generator. This recycling ultimately allows syntactic constraints to be localized, even though the grammar is not organized around English syntax.

Nitrogen's algorithm operates bottom-up, efficiently encoding multiple analyses in a lattice data structure to allow structure sharing, analogous to the way a chart is used in bottom-up parsing. In contrast, traditional generation control mechanisms work top-down, either deterministically (Meteer et al., 1987; Penman, 1989) or by backtracking to previous choice points (Elhadad, 1993b). This unnecessarily duplicates work at run time, unless sophisticated control directives are included in the search engine (Elhadad and Robin, 1992). Recently, (Kay, 1996) has explored a bottom-up approach to generation as well, using a chart rather than a word lattice.

Nitrogen's generation is robust and scalable. It can generate output even for unexpected or incomplete input, and is designed for broad coverage. It does not require the detailed, difficult-to-obtain knowledge bases that other NLG systems require, since it relies instead on corpus-based statistics to make a wide variety of linguistic decisions. Currently the quality of the output is limited by the use of only word bigram statistical information, which cannot handle long-distance agreement, or distinguish likely collocations from unlikely grammatical structure. However, we plan to remedy these problems by using statistical information extracted from the Penn Treebank corpus (Marcus et al., 1994) to rank tagged lattices and parse forests.

Nitrogen's rule matching is much less expensive than graph unification, and lattices generated for sub-AMRs are cached and reused in subsequent references. The semantic roles used in the grammar formalism cover most common syntactic phenomena, though our grammar does not yet generate questions, or infer pronouns from explicit coreference.

Nitrogen has been used extensively as part of a semantics-based Japanese-English MT system (Knight et al., 1995). Japanese analysis provides AMR's, which Nitrogen transforms into word lattices on the order of hundreds of nodes and thousands of arcs. These lattices compactly encode a number of syntactic variants that usually reach into the trillions and beyond. Most of these are somewhat ungrammatical or awkward, yet the statistical extractor rather successfully narrows them down to the top N best paths. An online demo is available at http://www.isi.edu/natural-language/mt/nitrogen/

References

J. Bateman. 1996. KPML development environment — multilingual linguistic resource development and sentence generation. Technical report, German Centre for Information Technology (GMD).

M. Elhadad and J. Robin. 1992. Controlling content realization with functional unification grammars. In R. Dale, E. Hovy, D. Roesner, and O. Stock, editors, *Aspects of Automated Natural Language Generation*. Springler Verlag.

M. Elhadad and J. Robin. 1998. Surge: a comprehensive plug-in syntactic realization component for text generation. In *http://www.cs.bgu.ac.il/research/projects/surge/*.

M. Elhadad. 1993a. FUF: The universal unifier—user manual, version 5.2. Technical Report CUCS-038-91, Columbia University.

M. Elhadad. 1993b. *Using Argumentation to Control Lexical Choice: A Unification-Based Implementation*. Ph.D. thesis, Columbia University.

M. Kay. 1996. Chart generation. In *Proc. ACL*.

K. Knight and V. Hatzivassiloglou. 1995. Two-level, many-paths generation. In *Proc. ACL*.

K. Knight and S. Luk. 1994. Building a large-scale knowledge base for machine translation. In *Proc. AAAI*.

K. Knight, I. Chander, M. Haines, V. Hatzivassiloglou, E. Hovy, M. Iida, S. K. Luk, R. Whitney, and K. Yamada. 1995. Filling knowledge gaps in a broad-coverage MT system. In *Proc. IJCAI*.

Benoit Lavoie and Owen Rambow. 1997. RealPro – a fast, portable sentence realizer. In *ANLP'97*.

M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*.

M. Meteer, D. McDonald, S. Anderson, D. Forster, L. Gay, A. Iluettner, and P. Sibun. 1987. Mumble-86: Design and implementation. Technical Report COINS 87-87, U. of Massachussets at Amherst, Amherst, MA.

G. Miller. 1990. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4). (Special Issue).

Penman. 1989. The Penman documentation. Technical report, USC/Information Sciences Institute.

Ehud Reiter. 1995. NLG vs. templates. In *Proc. ENLGW '95*.

S. Shieber, G. van Noord, R. Moore, and F. Pereira. 1989. A semantic-head-driven generation algorithm for unification based formalisms. In *Proc. ACL*.