Preserving Ambiguities in Generation via Automata Intersection

Kevin Knight and Irene Langkilde
Information Sciences Institute
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
{knight,ilangkil}@isi.edu

Abstract

We discuss the problem of generating text that preserves certain ambiguities, a capability that is useful in applications such as machine translation. We show that it is relatively simple to extend a hybrid symbolic/statistical generator to do ambiguity preservation. The paper gives algorithms and examples, and it discusses practical linguistic difficulties that arise in ambiguity preservation.

Introduction

This paper reports on an aspect of a natural language generation system called Nitrogen (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998ab; Langkilde, 2000), which has been used as part of the Gazelle machine translation (MT) system (Knight et al, 1995). In particular, we show how Nitrogen can generate English text that preserves unresolved ambiguities from non-English documents.

One of the first texts translated by Gazelle was a Japanese article with the following noun phrase:

1a. [IC chippu o seizou-suru noni tsukau]
dorai-echingu soochi ya suteppa
[*silicon chip construct for use*]
dry-etching device and stepper motor

In Japanese, relative clauses precede the nouns they modify. In this sentence, it is not clear whether the relative clause [shown bracketed] modifies the whole conjoined noun phrase (dry-etching device and stepper motor) or only the first noun in the conjunction (dry-etching device). Depending on the correct interpretation, reasonable English translations might be:

1b. ((dry-etching devices and stepper motors)
that are used to construct silicon chips)
1c. ((dry-etching devices that are used to construct
silicon chips) and stepper motors)

Both of these translations are "unsafe" because they commit to a particular interpretation that may turn out to be wrong. Translators who happen to know about computer equipment will prefer (1c) over (1b), but even (1c) has its problems—if we remove the parentheses, the sentence is likely to be misparsed by a reader!

The solution is to find a nice English translation that covers both interpretations:

1d. stepper motors and dry-etching devices
that are used to construct silicon chips

To get this "safe" translation, we simply re-order the nouns in the conjunction. It turns out that this conjunct-flipping technique is often employed by human translators. In this case, the idea is that the reader knows more about stepper motors than the translator does, and can disambiguate more easily. Many structural ambiguities provide opportunities for such ambiguity preservation.

Consider the English sentence:

2a. John saw the man with the telescope.

This sentence has two interpretations, depending on where the prepositional phrase attaches, but both interpretations are covered by the single Spanish translation:

2a. John vió al hombre con el telescopio.

This example shows how word-for-word MT engines frequently perform a simple-minded kind of ambiguity preservation, a fact that explains much of the commercial success of machine translation systems that operate between close language pairs. But even between Spanish and English, syntactic differences suggest careful handling. In the next phrase, the scope of the phrase-initial adjective is unclear:

3a. green eggs and ham

Again, conjunct flipping can help:

3b. (unsafe) huevos verdes y jamón
3c. (unsafe) huevos y jamón verdes
3d. (safe) jamón y huevos verdes

Another usefully vague construction is nominalization. Suppose that I witnessed somebody destroy something, and I only know that Nero was involved. There are safe and unsafe expressions for this situation:

4a. (unsafe) I witnessed Nero being destroyed.
4b. (unsafe) I witnessed the destruction by Nero.
4c. (safe) I witnessed Nero's destruction.

Pronoun reference can also benefit. Consider the English sentence

5. She saw the car in the window and wanted to buy it.

Although we must pick a gender for the German equivalent of "it," we can maintain the ambiguity if we select translations of "car" and "window" that have the same gender:

6a. (unsafe) Sie sah den Wagen$_{masc}$ im Schaufenster$_{neut}$ und wollte ihn$_{masc}$ kaufen.
6b. (unsafe) Sie sah den Wagen$_{masc}$ im Schaufenster$_{neut}$ und wollte es$_{neut}$ kaufen.
6c. (safe) Sie sah das Auto$_{neut}$ im Schaufenster$_{neut}$ und wollte es$_{neut}$ kaufen.

Opportunities for ambiguity preservation occur frequently, but most ambiguities cannot of course be preserved. For example, it is very difficult to preserve lexical part-of-speech ambiguities, as in sentences like "Time flies." Semantic lexical ambiguities are more frequently preservable. In the English sentence "I went to the center," it is not clear whether "center" means "middle" or "an institution devoted to the study of something." When we translate to Spanish, we can simply say "Fui al centro," without resolving the ambiguity.

Sometimes we can preserve an ambiguity only by appealing to awkward and disfluent constructions. Consider the following Japanese phrase:

7a. John no kuruma no kagi
John genitive-particle car genitive-particle key

There are two possible interpretations: ((John no kuruma) no kagi) and (John no (kuruma no kagi)). A translation like "the keys of John's car" commits to the first interpretation, while "John's car keys" commits to second. In this case, it doesn't really matter too much, but in other cases it will. So we would like to have a general method of preserving this kind of "X no Y no Z" ambiguity. Fortunately, there are two constructions that can do it—unfortunately, neither is very fluent English:

7b. (safe) the keys of the car of John
7c. (safe) John's car's keys

Another pitfall for ambiguity preservation is misdirection. A sentence that supposedly covers two interpretations may employ a syntactic structure that completely obscures one of them. For example, consider:

8. (supposedly safe) The CIA claimed that Jones was a spy from 1970 to 1992.

This sentence has a single overwhelmingly strong reading. The author of the sentence can always assert to the reader that it was actually the *claiming* that lasted for 1970 to 1992, not the *spying*, but in that case the reader will feel that he/she has been misdirected.

To sum up, ambiguity preservation is a technique often employed by human translators, among whom it is considered somewhat of an art. There are both opportunities and pitfalls. Ambiguity preservation is particularly interesting for machine translators, because they are not nearly as adept as humans at resolving source-language ambiguities.

Objective

Ideally, we would like to come up with an ambiguity preservation algorithm that is not tied to a specific language pair or even to specific linguistic constructions. Wedekind and Kaplan (1996) consider ambiguity preservation for generators that use LFG-style grammars. They show that the problem is undecidable: an interesting result, but one that is of little use to the practitioner. Emele and Dorna (1998) give an algorithm for preserving ambiguity that relies on packed LFG f-structure representations produced by the transfer component of an MT system. Shemtov (1998) describes the ambiguity preserving version of Kay's (1996) chart generator. Like the above authors, we will discuss ambiguity preservation across semantic inputs, thereby decoupling the problem from translation. For MT applications, we allow source language analysis to generate many possible meanings which are then picked over by an English-only generator. We also imagine non-MT applications for ambiguity preservation, for example, those that generate legal documents. Moreover, a generator that understands how to preserve ambiguity should also be able to steer clear of ambiguities, as again when generating legal documents.

We contribute ambiguity-preservation algorithma different from ones already proposed in (Shemtov, 1998; Emele and Dorna, 1998). Our algorithm uses structures that are somewhat simpler, and we are able to reduce our computations to well-known, generic operations on formal automata. Our goals are that (1) the method we invent is simple, (2) it is efficient, (3) it is a minimal extension to an existing generator, and (4) it is fully implemented and tested. We first describe the generator we use and then describe our method.

Nitrogen

Nitrogen is a broad-coverage sentence realizer that generates English from conceptual expressions that include all concepts of WordNet (Fellbaum, 1998) and a set of sixty semantic relationships drawn from the Gazelle MT interlingua. Because it is difficult to generate on a scale of 100,000 words and concepts without great deal of lexical, conceptual, and grammatical knowledge, Nitrogen supplements its shallow knowledge bases with statistical knowledge gathered from text. The statistical knowledge also helps it build fluent sentences from inputs that are underspecified with respect to number,

tense, definiteness, etc.

Nitrogen operates in two stages. First, a meaning representation is transformed into a large network of possible realizations, via an English grammar and lexicon. In the second stage, statistical knowledge is applied to select the most fluent realization.

In the first stage, it is the job of a grammar writer to to produce all possible realizations of input structures, i.e., particular configurations of semantic roles. This is much easier than specifying what are all the correct realizations in a myriad of sub-cases. The first stage of Nitrogen therefore massively overgenerates, producing a lot of ungrammatical and disfluent sentences. Most of these are weeded out by the second-stage statistical component.

Here is an example input to Nitrogen:

```
(p / possible,potential
  :domain (o / obligatory
             :domain (e / eat
                        :agent you
                        :patient chicken-meat)))
```

Nitrogen's grammar produces over ten million realizations, all packed into an efficient data structure. Here are a few randomly selected realizations:

9a. You may be obliged to eat that there was the poulet.
9b. A consumption of poulet by you may be the requirements.
9c. That the eating of chicken by you is obligatory is possible.

It is not trivial to say exactly what is wrong with such sentences, but they are clearly bad. Of the ten million possibilities, however, the statistical component ranks the following reasonable sentences as its top choices for output:

10a. You may have to eat chicken.
10b. You might have to eat chicken.
10c. You may be required to eat chicken.

We designed Nitrogen for scale, robustness, and accuracy, but it turns out that this set-up is good for ambiguity preservation as well.

Ambiguity Preservation with Word Lattices

The original version of Nitrogen packed its alternative realizations into a word-lattice data structure. A small fragment of sample Nitrogen word lattice is shown in Figure 1. The lattice has a start state, an end state, and transitions labeled with words. Each path corresponds to a different sentence, and the whole lattice can be viewed as a set of strings. Typical Nitrogen lattices have hundreds of nodes, thousands of arcs, and billions or more alternative sentences. To extract the most fluent sentence, we use word-pair statistics and n-best search procedures (Knight and Hatzivassiloglou, 1995) similar to those used in speech recognition.

Suppose we have two possible meaning representations (derived, say, from a single Spanish input sentence):

```
1.  (s / see
      :agent I
      :patient (m / man
                  :possesses (t / telescope)))

    -> I see the man with the telescope.
    -> I see the man holding the telescope.
            etc.

2.  (s / see
      :agent I
      :patient (m / man)
      :instrument (t / telescope))

    -> I see the man with the telescope.
    -> With the telescope, I see the man.
            etc.
```

Nitrogen can compute word lattices for these two meanings independently. Sentences that express both meanings simultaneously (i.e., preserve ambiguity) are exactly the sentences that occur in both lattices. Here, "I see the man with the telescope" appears in both, but "With the telescope, I see the man" does not.

We can compute the desired sentences simply by intersecting the two lattices. There are standard "book" algorithms for doing this. If we convert our lattices into deterministic finite-state acceptors FSA1 and FSA2, then Lewis and Papadimitriou (1981) give a simple algorithm that builds a new acceptor FSA3 which accepts the intersection of the strings accepted by FSA1 and FSA2. However, this algorithm is quadratic in the number of nodes, and is not very practical on large FSAs.

Ambiguity Preservation with Parse Forests

The current version of Nitrogen operates somewhat differently. It packs alternative realizations into a *parse-forest* structure.[1] While a word lattice is best viewed as a set of strings, a parse forest is best viewed as a set of trees. Figure 2 shows three sample forests.

We first designed and implemented an algorithm for intersecting a pair of parse forests, i.e., returning the set of syntactic trees that appear in both forests. While this algorithm preserves certain types of ambiguities, e.g., by conjunct-flipping and nominalization, it fails on others, such as the prepositional-phrase attachment example above. In a case like "I see the man with the

[1] Our main motivation is that in observing the behavior of Nitrogen's word-pair-based statistical ranking, we notice many errors due to missed long-distance dependencies. We believe that many of these errors will be corrected if we use a syntax-based ranking, such as that of Collins (1997) or Chelba and Jelinek (1998), operating over trees rather than flat strings. In order to match up with statistically-collected data, our trees are compliant with the labeling and bracketing scheme of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993). Another advantage of using forests is that they are more compact than lattices, avoiding repetition of substructures.

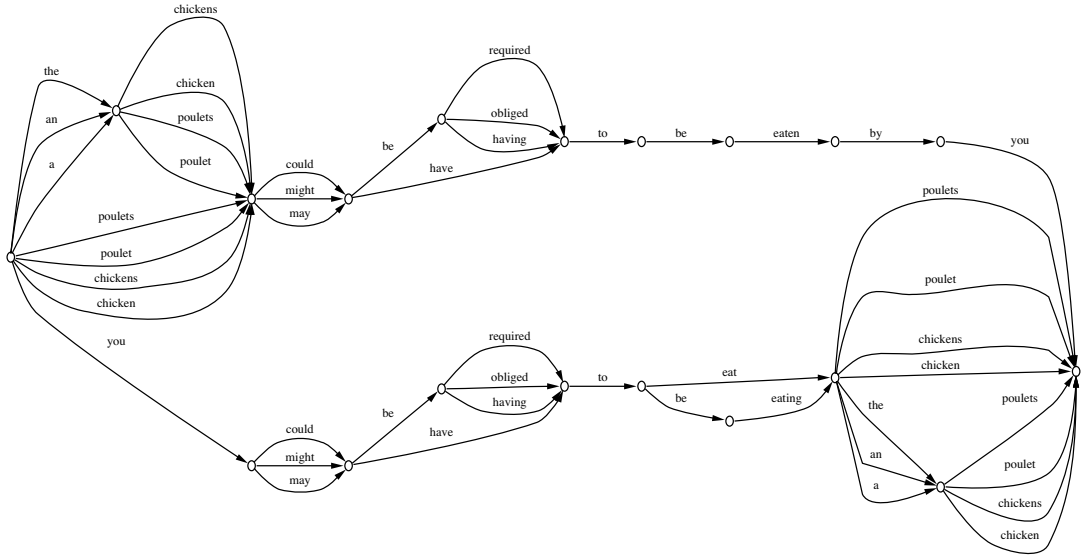

Figure 1: A small fragment of a sample Nitrogen word lattice. One path represents the fluent sentence *you may have to eat chicken*. Other paths correspond to less fluent or appropriate sentences. The full word lattice is many times larger than this fragment.

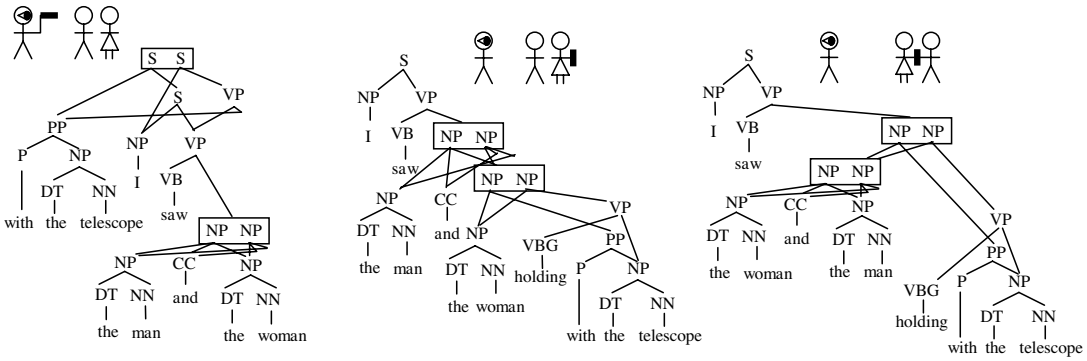

Figure 2: Three parse forests built from three semantic representations. The goal of ambiguity preservation is to identify the trees of just those sentences which cover all meanings.

telescope," there is no single syntactic tree that covers both meanings. However, there *are* strings that cover both meanings.

Suppose we are given parse forests PF1 and PF2. Then we want to build a new parse forest PF3 that includes every tree T such that either (1) T is in PF1 and T's terminal string is in PF2, or (2) T is in PF2 and T's terminal string is in PF1. The following algorithm can do this.

function *preserve-ambiguity*(PF1, PF2):

1. Create a lattice FSA1 that contains the same strings as the forest PF1, using the function shown in Figure 3. (The initial call to this recursive function should be *expand-forest-into-lattice*(PF1, root(PF1)).
2. Create a lattice FSA2 that contains the same strings as the forest PF2.
3. Write down PF1 in the form of a "grammar" composed of a number of context-free rewrite rules, e.g.:

TOP → S.1	SBAR.6 → NP.9 V.10
S.1 → NP.2 VP.3	NP.9 → *her*
NP.2 → *I*	V.10 → *move*
VP.3 → V.4 SBAR.6	NP.5 → ADJ.7 N.8
VP.3 → V.4 NP.5	ADJ.7 → *her*
V.4 → *saw*	N.8 → *move*

   This grammar derives only a finite number of strings.
4. Write down PF2 also using rewrite rules.
5. Compute the intersection of PF1 and FSA2. This results in a new forest that contains all trees in PF1 whose terminal strings are in FSA2. Bar Hillel (1961) describes an exponential-time algorithm for carrying out this intersection. As this is impractical, we instead adopt the efficient polynomial-time formulation

described by van Noord (1995):

> It can be shown that the computation of the intersection of a FSA and a CFG requires only a minimal generalization of existing parsing algorithms. We simply replace the usual string positions with the names of the states in the FSA. [p. 160]

Instead of using the PF1 "grammar" to parse a string, we use it to parse the lattice FSA2. We adopt a standard chart parser (Allen, 1989), replacing string positions with lattice states. We process the lattice transitions of FSA2 from left to right (in topological order), adding new constituents to the chart and packing ambiguities in the standard way. This kind of lattice parsing is frequently used in parsing noisy input, as from speech recognition.

6. Compute the intersection of PF2 and FSA1.
7. Compute the union of the resulting forests (results of steps 5 and 6) by merging their root nodes.

To preserve the three-way ambiguity of forests A, B, and C in Figure 2, we apply the above algorithm twice, i.e., *preserve-ambiguity*(*preserve-ambiguity*(A, B), C).

We have implemented this algorithm and integrated it into the Nitrogen generator. When Nitrogen encounters a disjunction in its semantic input, it invokes the ambiguity preservation routine. We have observed this routine employing several different syntactic devices such as those we described in the first section.

* * *

Due to Nitrogen's implementation there is a more efficient algorithm than the one just described for computing ambiguity-preserving trees. This algorithm is based on two aspects of Nitrogen's implementation. First, Nitrogen uses a cache to store the results of mapping sub-pieces of the input to sub-forests. The cache avoids duplicate processing during generation because if a particular sub-piece of an input occurs in multiple inputs, the result can retrieved from the cache rather than being regenerated. The effect of the cache is that different forests will share sub-trees if they have sub-units of semantic representation in common. Such shared trees inherently represent an intersection between forests, and automatically preserve ambiguity between different inputs. Unnecessary processing can be avoided by recognizing this in the ambiguity preserving algorithm, rather than continuing to search within the shared subtree.

The second aspect of Nitrogen's implementation that facilitates the preservation of ambiguities is the labeling of nodes in the tree. When a sub-forest is generated, it is identified with a unique numeric label. Successive sub-forests are assigned numerically increasing labels. A side effect of this labeling is that a partial order on the nodes in forest is guaranteed, such that a parent node always has a higher-numbered label than any of its children. The ordering on the nodes can be

```
function expand-forest-into-lattice(forest, node)
. lat ← empty-lattice()
. for each rule r in forest with lhs(r) = symbol(node):
. . lat2 ← empty-lattice()
. . for each y in rhs(r):
. . . lat3 ← expand-forest-into-lattice(forest, y)
. . . merge-lattice-states(final-state(lat2), start-state(lat3))
. . merge-lattice-states(start-state(lat), start-state(lat2))
. . merge-lattice-states(final-state(lat), final-state(lat2))
. return(lat)
```

Figure 3: Expanding a forest into a lattice.

```
function tree-intersect (A B)
if A and B are equal then return the pair (A,B);
if there are no constituents in A or B then return nil;
if the highest node of A OR the highest node of B is a leaf node,
. then return nil;
if the highest nodes of A and B are the same then
. high-node := highest node of A;
. res1 := tree-intersect(left-of-highest(A),left-of-highest(B));
. res2 := tree-intersect(right-of-highest(A),right-of-highest(B));
. for every tree pair r1 in res1
. . for every tree pair r2 in res2
. . . A1 := append the A tree of res1 to the left of high-node
. . . . and the A tree of res2 to the right of the high node;
. . . B1 := append the B tree of res1 to the left of high-node
. . . . and the B tree of res2 to the right of the high node;
. . . add the pair (A1,B1) to the tree-pair list;
. return the tree-pair list.
if the highest node of A is greater than the highest node of B,
. for each disjunctive set S of children of A
. . A1 := substitute S for high-node in A
. . res := tree-intersect(A1,B);
. for every tree pair r in res
. . A2 := replace children in former positions of S
. . . with new parent node
. return list of tree-pairs (A2,B);
otherwise, do the previous seven steps
. reversing the roles of A and B.
```

Figure 4: Tree-Intersection Algorithm

used to guide the search for an intersection between forests. Higher-numbered nodes are always expanded first, ensuring that a shared-subtree will be discovered, if it exists, without doing any unnecessary processing. When a shared sub-tree is found and it is the highest-numbered node among the nodes being examined, the divide-and-conquer method can be used to split the remaining search into two independent sub-problems, thereby significantly reducing the overall search space.

The algorithm in Figure 4 exploits these two aspects of Nitrogen's implementation to achieve more efficient processing. The A and B arguments to the *tree-intersect* function are lists of sequential nodes. Initially, each list contains only the root node of the respective trees being compared. The algorithm is not limited to comparing whole forests, but equally well handles sub-forests and arbitrary sequences of sibling nodes. This gives it another advantage over the previous described algorithm since it can be integrated into the genera-

tion process, permitting the preservation of ambiguities within inputs as well as across inputs. The algorithm returns a list of paired trees sharing the same sequence of words, with their differing syntactic structure preserved. This algorithm has been implemented and integrated into the Nitrogen generator.

Discussion

It was very easy to adapt Nitrogen to perform a wide range of ambiguity preservations. Partly, this is because of the simple lattice and forest representations it uses. Nitrogen dispenses with the feature notations that bedevil Wedekind's generator, Shemtov's ambiguity preserving algorithm, and van Noord's lattice parser. In our experience, statistically-gathered knowledge can simulate these features fairly well, while also taking into account the collocational properties of language that are very hard to model with features.

If an ambiguity is not preservable, we union the parse forests instead of doing intersections. This amounts to letting the statistical ranker choose the most fluent sentence regardless of which meaning it expresses. This works better than expected, because strange meanings often make for strange sentences. We do not yet address the problems of disfluent preservations or misdirection (examples 7 and 8), but there are promising avenues in the Nitrogen framework. For example, we might do both intersection and union, but assign higher weights to trees in the intersection. These weights would multiply with those of the statistical ranker, allowing a non-preserving fluent phrase ("John's car keys") to overcome a preserving disfluent one ("John's car's keys"). Misdirection might be attacked by looking at the probabilities assigned to trees by a syntax-based statistical ranker, or by examining the top n statistical parses of proposed generator output.

We would also like to adopt a flat-structure representation for underspecified semantics and massive ambiguity packing (Copestake et al, 1995; Alshawi, 1996; Reyle, 1996, Doerre, 1997), which has been useful to (Shemtov, 1998; Emele and Dorna, 1998). Nitrogen's current meaning representation language, while allowing local disjunctions, is a straightforward but limited feature-based one.

Acknowledgments

We gratefully acknowledge support from NSF Award IIS-9820291.

References

Allen, J. (1989). *Natural Language Understanding*, Benjamin/Cummings.

Alshawi, H. (1996). Underspecified First Order Logics, in Semantic Ambiguity and Underspecification, eds. K. van Deemter and S. Peters.

Bar-Hillel, Y., M. Perles, and E. Shamir (1961). On Formal Properties of Simple for a Structure Grammars, Zeitschrift für Phonetik, SprachWissenschaft und Kommunicationsforschung, 14:143- 172, 1961. Reprinted in *Language and Information - Selected Essays on their Theory and Application*, Addison Wesley series in Logic, 1964, pp. 116-150.

Chelba, C. and F. Jelinek (1998). Exploiting Syntactic Structure for Language Modeling, Proc. COLING/ACL.

Collins, M. (1997). Three Generative, Lexicalised Models for Statistical Parsing, Proc. ACL.

Copestake, A., D. Flickinger, R. Maloufand, S. Riehemann, and I. Sag (1995). Translation using Minimal Recursion Semantics, Proc. Theoretical and Methodological Issues in Machine Translation (TMI).

Doerre, J. (1997). Efficient Construction of Underspecified Semantics under Massive Ambiguity, Proc. ACL/EACL.

Emele, M. and M. Dorna (1998). Ambiguity Preserving Machine Translation using Packed Representations, Proc. COLING/ACL.

Fellbaum, C. (1998). *WordNet*, MIT Press. Kay, M. (1996).

Chart Generation, Proc. ACL.

Knight, K., I. Chander, M. Haines, V. Hatzivassiloglou, E. Hovy, M. Iida, S. Luk, R. Whitney, K. Yamada (1995). Filling Knowledge Gaps in a Broad-Coverage MT System, Proc. IJCAI.

Knight, K. and V. Hatzivassiloglou (1995). Two-Level, Many-Paths Generation, Proc. ACL.

Langkilde, I. Forest-Based Statistical Sentence Generation, Proc. North American ACL.

Langkilde, I. and K. Knight (1998a), The Practical Value of N-Grams in Generation, Proc. of the International Workshop on Natural Language Generation.

Langkilde, I. and K. Knight (1998b), Generation that Exploits Corpus-Based Statistical Knowledge, Proc. COLING/ACL.

Lewis, H. and C. Papadimitriou (1981). *Elements of the Theory of Computation*, Prentice-Hall, Inc.

Marcus, M., B. Santorini, and M. Marcinkiewicz (1993). Building a Large Annotated Corpus of English: The Penn Treebank, *Computational Linguistics*, 19(2).

by Composition of

Reyle, U. (1996). Co-indexing Labelled DRSs to Represent and Reason with Ambiguities, in *Semantic Ambiguity and Underspecification*, eds. K. van Deemter and S. Peters.

Shemtov, H. (1998). *Ambiguity Management in Natural Language Generation*, Ph.D. thesis, Department of Linguistics, Stanford University.

van Noord, G. (1995). The Intersection of Finite State Automata and Definite Clause Grammars, Proc. ACL.

Wedekind, J. and R. Kaplan (1996). Ambiguity-Preserving Generation with LFG- and PATR-style Grammars, *Computational Linguistics*, 22(4).