Proc. 2002 Conf. on Empirical Methods in Natural Language Processing (EMNLP)

Bootstrapping Lexical Choice via Multiple-Sequence Alignment

Regina Barzilay
Department of Computer Science
Columbia University
New York, NY 10027 USA
regina@cs.columbia.edu

Lillian Lee
Department of Computer Science
Cornell University
Ithaca, NY 14853 USA
llee@cs.cornell.edu

Abstract

An important component of any generation system is the *mapping dictionary*, a lexicon of elementary semantic expressions and corresponding natural language realizations. Typically, labor-intensive knowledge-based methods are used to construct the dictionary. We instead propose to acquire it automatically via a novel multiple-pass algorithm employing *multiple-sequence alignment*, a technique commonly used in bioinformatics. Crucially, our method leverages latent information contained in *multi-parallel* corpora — datasets that supply several verbalizations of the corresponding semantics rather than just one.

We used our techniques to generate natural language versions of computer-generated mathematical proofs, with good results on both a per-component and overall-output basis. For example, in evaluations involving a dozen human judges, our system produced output whose readability and faithfulness to the semantic input rivaled that of a traditional generation system.

1 Introduction

One or two homologous sequences whisper . . . a full multiple alignment shouts out loud (Hubbard et al., 1996).

Today's natural language generation systems typically employ a *lexical chooser* that translates complex semantic concepts into words. The lexical chooser relies on a *mapping dictionary* that lists possible realizations of elementary semantic concepts; sample entries might be `[Parent [sex:female]]` → MOTHER or `love(`x`,`y`)`→ $\{x$ LOVES y, x IS IN LOVE WITH $y\}$.[1]

To date, creating these dictionaries has involved human analysis of a domain-relevant corpus comprised of semantic representations and corresponding human verbalizations (Reiter and Dale, 2000). The corpus analysis and knowledge engineering work required in such an approach is substantial, prohibitively so in large domains. But, since corpus data is already used in building lexical choosers by hand, an appealing alternative is to have the system learn a mapping dictionary directly from the data. Clearly, this would greatly reduce the human effort involved and ease porting the system to new domains. Hence, we address the following problem: given a parallel (but unaligned) corpus consisting of both complex semantic input and corresponding natural language verbalizations, learn a semantics-to-words mapping dictionary automatically.

Now, we could simply apply standard statistical machine translation methods, treating verbalizations as "translations" of the semantics. These methods typically rely on *one-parallel* corpora consisting of text pairs, one in each "language" (but cf. Simard (1999); see Section 5). However, learning the kind of semantics-to-words mapping that we desire from one-parallel data alone is difficult even for humans. First, given the same semantic input, different authors may (and do) delete or insert information (see Figure 1); hence, direct comparison between a semantic text and a single verbalization may not provide enough information regarding their underlying correspondences. Second, a single verbalization certainly fails to convey the variety of potential linguistic realizations of the concept that an expressive lexical chooser would ideally have access to.

The multiple-sequence idea Our approach is motivated by an analogous situation that arises in computational biology. In brief, an important bioin-

[1] Throughout, fonts denote a mapping dictionary's two information types: `semantics` and REALIZATIONS.

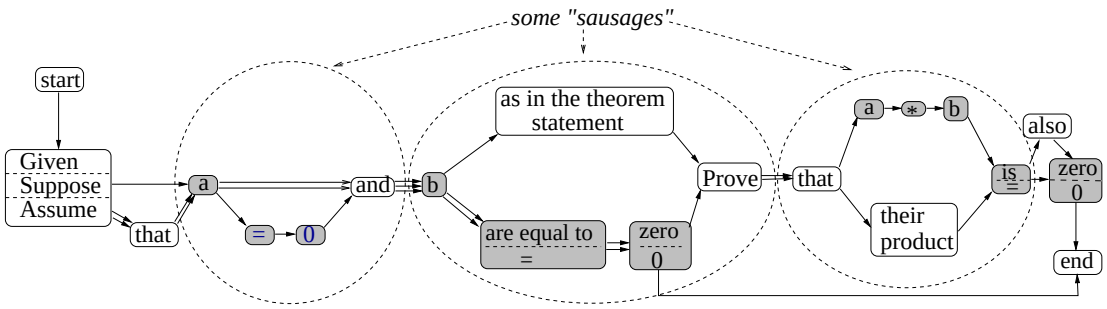

Figure 2: Computed lattice for verbalizations from Figure 1. Note how the three indicated "sausages" roughly correspond to the three arguments of `show-from(a=0,b=0,a*b=0)`. (The phrases "as in the theorem statement" and "their product" correspond to chains of nodes, but are drawn as single nodes for clarity. Shading indicates argument-value matches (Section 3.1). All lattice figures omit punctuation nodes for clarity.)

(1) Given a and b as in the theorem statement, prove that a∗b=0.
(2) Suppose that a and b are equal to zero. Prove that their product is also zero.
(3) Assume that a=0 and b=0.

Figure 1: Three different human verbalizations of `show-from(a=0,b=0,a*b=0)`.

formatics problem — Gusfield (1997) refers to it as "The Holy Grail" — is to determine commonalities within a collection of biological sequences such as proteins or genes. Because of mutations within individual sequences, such as changes, insertions, or deletions, pair-wise comparison of sequences can fail to reveal which features are conserved across the entire group. Hence, biologists compare multiple sequences simultaneously to reveal hidden structure characteristic to the group as a whole.

Our work applies multiple-sequence alignment techniques to the mapping-dictionary acquisition problem. The main idea is that using a *multi-parallel corpus* — one that supplies *several* alternative verbalizations for each semantic expression — can enhance both the accuracy and the expressiveness of the resulting dictionary. In particular, matching a semantic expression against a composite of the common structural features of a set of verbalizations ameliorates the effect of "mutations" within individual verbalizations. Furthermore, the existence of multiple verbalizations helps the system learn several ways to express concepts.

To illustrate, consider a sample semantic expression from the mathematical theorem-proving domain. The expression `show-from(a=0,b=0,a*b=0)` means "assuming the two premises $a = 0$ and $b = 0$, show that the goal $a * b = 0$ holds". Figure 1 shows three human verbalizations of this expression. Even for so formal a domain as mathematics, the verbalizations vary considerably, and none directly matches the entire semantic input. For instance, it is not obvious without domain knowledge that "Given a and b as in the theorem statement" matches "`a=0`" and "`b=0`", nor that "their product" and "`a*b`" are equivalent. Moreover, sentence (3) omits the goal argument entirely. However, as Figure 2 shows, the combination of these verbalizations, as computed by our multiple-sequence alignment method, exhibits high structural similarity to the semantic input: the indicated "sausage" structures correspond closely to the three arguments of `show-from`.

2 Multiple-sequence alignment

This section describes general multiple-sequence alignment; we discuss its application to learning mapping dictionaries in the next section.

A multiple-sequence alignment algorithm takes as input n strings and outputs an n-row correspondence table, or *multiple-sequence alignment* (MSA). (We explain how the correspondences are actually computed below.) The MSA's rows correspond to sequences, and each column indicates which elements of which strings are considered to correspond at that point; non-correspondences, or "gaps", are represented by underscores (_). See Figure 3(i).

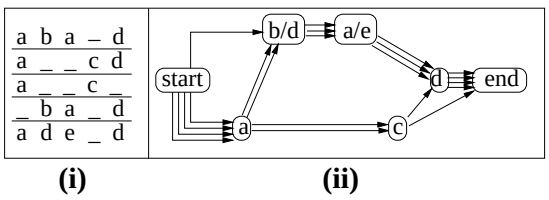

Figure 3: (i) An MSA of five sequences (the first is "abad"); (ii) The corresponding lattice.

From an MSA, we can compute a *lattice* . Each lattice node, except for "start" and "end", corre-

sponds to an MSA column. The edges are induced by traversing each of the MSA's rows from left to right. See Figure 3(ii).

Alignment computation The sum-of-pairs dynamic programming algorithm and pairwise iterative alignment algorithm sketched here are described in full in Gusfield (1997) and Durbin et al. (1998).

Let Σ be the set of elements making up the sequences to be aligned, and let $\text{sim}(x, y)$, x and $y \in \Sigma \cup \{_\}$, be a domain-specific *similarity function* that assigns a score to every possible pair of alignment elements, including gaps. Intuitively, we prefer MSAs in which many high-similarity elements are aligned.

In principle, we can use dynamic programming over alignments of sequence prefixes to compute the highest-scoring MSA, where the *sum-of-pairs* score for an MSA is computed by summing $\text{sim}(x, y)$ over each pair of entries in each column. Unfortunately, these computations are exponential in n, the number of sequences. (In fact, finding the optimal MSA when n is a variable is NP-complete (Wang and Jiang, 1994).) Therefore, we use *iterative pairwise* alignment, a commonly-used polynomial-time approximation procedure, instead. This algorithm greedily merges pairs of MSAs of (increasingly larger) *subsets* of the n sequences; which pair to merge is determined by the average score of all pairwise alignments of sequences from the two MSAs.

Aligning lattices We can apply the above sequence alignment algorithm to lattices as well as sequences, as is indeed required by pairwise iterative alignment. We simply treat each lattice as a sequence whose ith symbol corresponds to the set of nodes at distance i from the start node. We modify the similarity function accordingly: any two new symbols are equivalent to subsets S_1 and S_2 of Σ, so we define the similarity of these two symbols as $\max_{(x,y) \in S_1 \times S_2} \text{sim}(x, y)$.

3 Dictionary Induction

Our goal is to produce a semantics-to-words mapping dictionary by comparing semantic sequences to MSAs of multiple verbalizations. We assume only that the semantic representation uses predicate-argument structure, so the elementary semantic units are either *terms* (e.g., `0`), or *predicates* taking arguments (e.g., `show-from`$(prem1, prem2, goal)$, whose arguments are two premises and a goal). Note that both types of units can be verbalized by multi-word sequences.

Now, semantic units can occur several times in the corpus. In the case of predicates, we would like to combine information about a given predicate from *all* its appearances, because doing so would yield more data for us to learn how to express it. On the other hand, correlating verbalizations across instances instantiated with different argument values (e.g., `show-from(a=0,b=0,a*b=0)` vs. `show-from(c>0,d>0,c/d>0)`) makes alignment harder, since there are fewer obvious matches (e.g., "a∗b=0" does not greatly resemble "c/d>0"); this seems to discourage aligning cross-instance verbalizations.

We resolve this apparent paradox by a novel three-phase approach:

- In the *per-instance alignment* phase (Section 3.1), we handle each separate instance of a semantic predicate individually. First, we compute a separate MSA for each instance's verbalizations. Then, we abstract away from the particular argument values of each instance by replacing lattice portions corresponding to argument values with *argument slots*, thereby creating a *slotted lattice*.
- In the *cross-instance alignment* phase (Section 3.2), for each predicate we align together all the slotted lattices from *all* of its instances.
- In the *template induction* phase (Section 3.3), we convert the aligned slotted lattices into *templates* — sequences of words and argument positions — by tracing slotted lattice paths.

Finally, we enter the templates into the mapping dictionary.

3.1 Per-instance alignment

As mentioned above, the first job of the per-instance alignment phase is to separately compute for each instance of a semantic unit an MSA of all its verbalizations. To do so, we need to supply a scoring function capturing the similarity in meaning between words. Since such similarity can be domain-dependent, we use the data to induce — again via sequence alignment — a *paraphrase thesaurus* T that lists linguistic items with similar meanings. (This process is described later in section 3.1.1.) We then set

$$\text{sim}(x, y) = \begin{cases} 1 & x = y,\ x \in \Sigma; \\ 0.5 & x \approx y; \\ -0.01 & \text{exactly one of } x, y \text{ is } _\ ; \\ -0.5 & \text{otherwise (mismatch)} \end{cases}$$

where Σ is the vocabulary and $x \approx y$ denotes that T lists x and y as paraphrases.[2] Figure 2 shows the lattice computed for the verbalizations of the instance

[2]These values were hand-tuned on a held-out development corpus, described later. Because we use progressive alignment, the case $x = y = _$ does not occur.

`show-from(a=0,b=0,a*b=0)` listed in Figure 1. The structure of the lattice reveals why we informally refer to lattices as "sausage graphs".

Next, we transform the lattices into slotted lattices. We use a simple matching process that finds, for each argument value in the semantic expression, a sequence of lattice nodes such that each node contains a word identical to or a paraphrase of (according to the paraphrase thesaurus) a symbol in the argument value (these nodes are shaded in Figure 2). The sequences so identified are replaced with a "slot" marked with the argument variable (see Figure 4).[3] Notice that by replacing the argument values with variable labels, we make the commonalities between slotted lattices for different instances more clear, which is useful for the cross-instance alignment step.

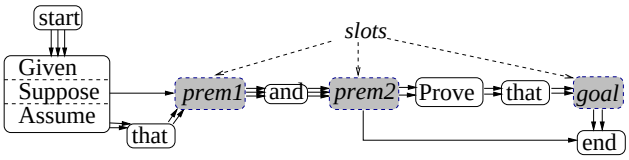

Figure 4: Slotted lattice, computed from the lattice in Figure 2, for `show-from`$(prem1, prem2, goal)$.

3.1.1 Paraphrase thesaurus creation

Recall that the paraphrase thesaurus plays a role both in aligning verbalizations and in matching lattice nodes to semantic argument values. The main idea behind our paraphrase thesaurus induction method, motivated by Barzilay and McKeown (2001), is that paths through lattice "sausages" often correspond to alternate verbalizations of the same concept, since the sausage endpoints are contexts common to all the sausage-interior paths. Hence, to extract paraphrases, we first compute all pairwise alignments of parallel verbalizations, discarding those of score less than four in order to eliminate spurious matches.[4] Parallel sausage-interior paths that appear in several alignments are recorded as paraphrases. Then, we iterate, realigning each pair of sentences, but with previously-recognized paraphrases treated as identical, until no new paraphrases are discovered. While the majority of the derived paraphrases are single words, the algorithm also produces several multi-word paraphrases, such as "are equal to" for "=". To simplify subsequent comparisons, these phrases (e.g., "are equal to") are treated as single tokens. Here are four paraphrase pairs we extracted from the mathematical-proof domain:

(conclusion, result)	(0, zero)
(applying, by)	(expanding, unfolding)

(See Section 4.2 for a formal evaluation of the paraphrases.) We treat thesaurus entries as degenerate slotted lattices containing no slots; hence, terms and predicates are represented in the same way.

3.2 Cross-instance alignment

Figure 4 is an example where the verbalizations for a single instance yield good information as to how to realize a predicate. (For example, "ASSUME *[prem1]* AND *[prem2]*, PROVE *[goal]*", where the brackets enclose arguments marked with their type.) Sometimes, though, the situation is more complicated. Figure 5 shows two slotted lattices for different instances of `rewrite`$(lemma, goal)$ (meaning, rewrite *goal* by applying *lemma*); the first slotted lattice is problematic because it contains context-dependent information (see caption). Hence, we engage in *cross-instance alignment* to merge information about the predicate. That is, we align the slotted lattices for *all* instances of the predicate (see Figure 6); the resultant *unified slotted lattice* reveals linguistic expressions common to verbalizations of different instances. Notice that the argument-matching process in the per-instance alignment phase helps make these commonalities more evident by abstracting over different values of the same argument (e.g., `lemma100` and `lemma104` are both relabeled "*lemma*").

3.3 Template induction

Finally, it remains to create the mapping dictionary from unified slotted lattices. While several strategies are possible, we chose a simple *consensus sequence* method. Define the *node weight* of a given slotted lattice node as the number of verbalization paths passing through it (downweighted if it contains punctuation or the words "the", "a", "to", "and", or "of"). The *path weight* of a slotted lattice path is a length-normalized sum of the weights of its nodes.[5] We produce as a template the words from the *consensus sequence*, defined as the maximum-weight path, which is easily computed via dynamic programming. For example, the template we derive from Figure 6's slotted lattice is WE USE LEMMA [*lemma*] TO GET [*goal*].

[3] This may further change the topology by forcing other nodes to be removed as well. For example, the slotted lattice in Figure 4 doesn't contain the node sequence "their product".

[4] Pairwise alignments yield fewer candidate alignments from which to select paraphrases, allowing simple scoring functions to produce decent results.

[5] Shorter paths are preferred, but we discard sequences shorter than six words as potentially spurious.

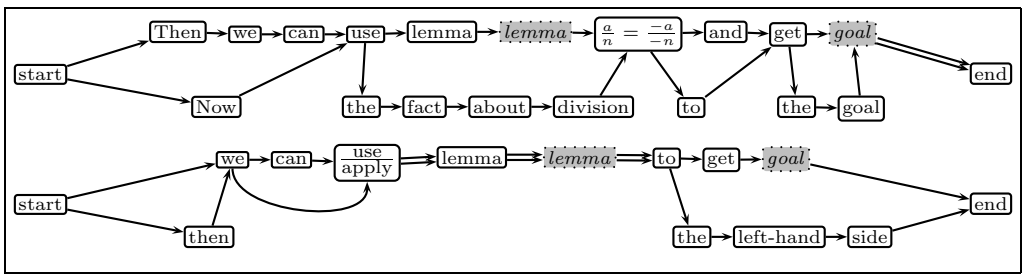

Figure 5: Slotted lattices for the predicate `rewrite`*(lemma,goal)* derived from two instances:
(instance I) `rewrite(lemma100,a-n*((-a)/(-n))=-(-a-(-n)*((-a)/(-n))))`, and
(instance II) `rewrite(lemma104,A-(-(A/(-N)))*N = A-(A/(-N))*(-N))`;
each instance had two verbalizations. In instance (I), both verbalizations contain the context-dependent information "$\frac{a}{n} = \frac{-a}{-n}$" (the statement of `lemma100`); also, argument-matching failed on the context-dependent phrase "the fact about division".

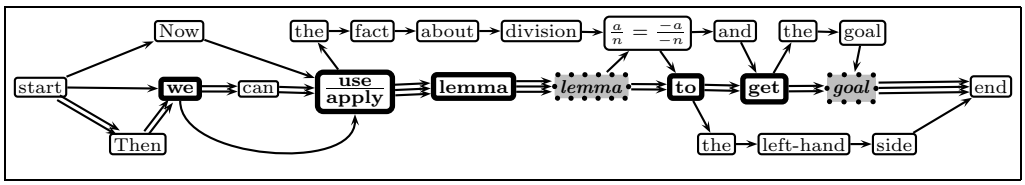

Figure 6: Unified slotted lattice computed by cross-instance alignment of Figure 5's slotted lattices. The consensus sequence is shown in bold (recall that node weight roughly corresponds to in-degree).

While this method is quite efficient, it does not fully exploit the expressive power of the lattice, which may encapsulate several valid realizations. We leave to future work experimenting with alternative template-induction techniques; see Section 5.

4 Evaluation

We implemented our system on formal mathematical proofs created by the Nuprl system, which has been used to create thousands of proofs in many mathematical fields (Constable et al., 1986). Generating natural-language versions of proofs was first addressed several decades ago (Chester, 1976). But now, large formal-mathematics libraries are available on-line.[6] Unfortunately, they are usually encoded in highly technical languages (see Figure 7(i)). Natural-language versions of these proofs would make them more widely accessible, both to users lacking familiarity with a specific prover's language, and to search engines which at present cannot search the symbolic language of formal proofs.

Besides these practical benefits, the formal mathematics domain has the further advantage of being particularly suitable for applying statistical generation techniques. Training data is available because theorem-prover developers frequently provide verbalizations of system outputs for explanatory purposes. In our case, a multi-parallel corpus of Nuprl proof verbalizations already exists (Holland-Minkley et al., 1999) and forms the core of our training corpus. Also, from a research point of view, the examples from Figure 1 show that there is a surprising variety in the data, making the problem quite challenging.

All evaluations reported here involved judgments from graduate students and researchers in computer science. We authors were not among the judges.

4.1 Corpus

Our training corpus consists of 30 Nuprl proofs and 83 verbalizations. On average, each proof consists of 5.08 *proof steps*, which are the basic semantic unit in Nuprl; Figure 7(i) shows an example of three Nuprl steps. An additional five proofs, disjoint from the test data, were used as a development set for setting the values of all parameters.[7]

Pre-processing First, we need to divide the verbalization texts into portions corresponding to individual proof steps, since per-instance alignment handles verbalizations for only one semantic unit at a time. Fortunately, Holland-Minkley et al. (1999)

[6] See `http://www.cs.cornell.edu/Info/Projects/-NuPrl/` or `http://www.mizar.org`, for example.

[7] See `http://www.cs.cornell.edu/Info/Projects/NuPrl/html/nlp` for all our data.

(i)	(ii)	(iii)

`UnivCD(`$\forall$ `i:`$\mathbb{N}$`.`$\lvert$`i`$\rvert = \lvert$`-i`$\rvert$`, i:`$\mathbb{N}$, $\lvert$`i`$\rvert = \lvert$`-i`$\rvert$`)`
`BackThruLemma(`$\lvert$`i`$\rvert = \lvert$`-i`$\rvert$, `i=` $\pm$ `i,absval_eq)`
`Unfold(i=` $\pm$ `i, (), pm_equal)`
	Assume that i is an integer, we need to show $\lvert i\rvert = \lvert -i\rvert$. From `absval_eq` lemma, $\lvert i\rvert = \lvert -i\rvert$ reduces to $i = \pm i$. By the definition of `pm_equal`, $i = \pm i$ is proved.	Assume i is an integer. By the `absval_eq` lemma, the goal becomes $\lvert i\rvert = \lvert -i\rvert$. Now, the original expression can be rewritten as $i = \pm i$.

Figure 7: (i) Nuprl proof (test lemma "h" in Figure 8). (ii) Verbalization produced by our system. (iii) Verbalization produced by traditional generation system; note that the initial goal is never specified, which means that in the phrase "the goal becomes", we don't know what the goal is.

showed that for Nuprl, one proof step roughly corresponds to one sentence in a natural language verbalization. So, we align Nuprl steps with verbalization sentences using dynamic programming based on the number of symbols common to both the step and the verbalization. This produced 382 pairs of Nuprl steps and corresponding verbalizations. We also did some manual cleaning on the training data to reduce noise for subsequent stages.[8]

4.2 Per-component evaluation

We first evaluated three individual components of our system: paraphrase thesaurus induction, argument-value selection in slotted lattice induction, and template induction. We also validated the utility of multi-parallel, as opposed to one-parallel, data.

Paraphrase thesaurus We presented two judges with all 71 paraphrase pairs produced by our system. They identified 87% and 82%, respectively, as being plausible substitutes within a mathematical context.

Argument-value selection We next measured how well our system matches semantic argument values with lattice node sequences. We randomly selected 20 Nuprl steps and their corresponding verbalizations. From this sample, a Nuprl expert identified the argument values that appeared in at least one corresponding verbalization; of the 46 such values, our system correctly matched lattice node sequences to 91%. To study the relative effectiveness of using multi-parallel rather than one-parallel data, we also implemented a baseline system that used only *one* (randomly-selected) verbalization among the multiple possibilities. This single-verbalization baseline matched only 44% of the values correctly, indicating the value of a multi-parallel-corpus approach.

Templates Thirdly, we randomly selected 20 induced templates; of these, a Nuprl expert determined that 85% were plausible verbalizations of the corresponding Nuprl. This was a very large improvement over the single-verbalization baseline's 30%, again validating the multi-parallel-corpus approach.

4.3 Evaluation of the generated texts

Finally, we evaluated the quality of the text our system generates by comparing its output against the system of Holland-Minkley et al. (1999), which produces accurate and readable Nuprl proof verbalizations. Their system has a hand-crafted lexical chooser derived via manual analysis of the same corpus that our system was trained on. To run the experiments, we replaced Holland-Minkley et. al's lexical chooser with the mapping dictionary we induced. (An alternative evaluation would have been to compare our output with human-authored texts. But this wouldn't have allowed us to evaluate the performance of the lexical chooser alone, as human proof generation may differ in aspects other than lexical choice.) The test set serving as input to the two systems consisted of 20 held-out proofs, unseen throughout the entirety of our algorithm development work. We evaluated the texts on two dimensions: readability and fidelity to the mathematical semantics.

Readability We asked 11 judges to compare the readability of the texts produced from the same Nuprl proof input: Figure 7(ii) and (iii) show an example text pair.[9] (The judges were not given the original Nuprl proofs.) Figure 8 shows the results. *Good* entries are those that are *not* preferences for the traditional system, since our goal, after all, is to show that MSA-based techniques can produce output as good or better than a hand-crafted system. We see that for *every* lemma and for *every* judge, our system performed quite well. Furthermore, for more than half of the lemmas, more than half the

[8] We employed pattern-matching tools to fix incorrect sentence boundaries, converted non-ascii symbols to a human-readable format, and discarded a few verbalizations which were unrelated to the underlying proof.

[9] To prevent judges from identifying the system producing the text, the order of presentation of the two systems' output was randomly chosen anew for each proof.

Judge	Lemma																				% good
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	
A	■	■	■	■	♦	■	■	■	■	■	■	♦	■	■	■	■	■	♦	■	♦	100
B	□	□	■	□	■	■	♦	■	■	♦	♦	♦	□	■	■	■	■	■	□	■	75
C	♦	♦	□	□	♦	■	□	■	□	□	■	♦	♦	■	■	♦	□	■	♦	♦	70
D	□	■	■	♦	♦	♦	■	□	■	□	□	■	♦	□	■	■	□	■	♦	■	70
E	□	♦	■	♦	♦	♦	■	♦	■	□	□	♦	♦	□	♦	□	■	♦	□	♦	70
F	■	■	■	■	□	♦	□	■	■	■	■	■	■	■	□	■	■	■	■	■	85
G	□	■	■	■	♦	□	■	■	♦	■	■	♦	■	■	□	♦	♦	■	♦	♦	85
H	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100
I	□	□	■	♦	♦	□	□	■	■	□	■	□	□	■	♦	♦	■	■	■	□	60
J	♦	♦	■	♦	■	♦	♦	■	♦	♦	♦	♦	■	■	■	■	□	□	□	■	85
K	■	■	■	□	■	■	■	□	□	♦	□	□	■	■	■	■	□	□	♦	■	65
% good	55	82	91	73	91	82	73	82	82	64	73	82	82	82	82	91	64	82	73	91	
> 50% ■?		✓	✓				✓	✓	✓		✓		✓	✓	✓	✓	✓	✓		✓	

Figure 8: Readability results. ■: preference for our system. □: preference for hand-crafted system. ♦: no preference. ✓: > 50% of the judges preferred the statistical system's output.

judges found our system's output to be distinctly better than the traditional system's.

Fidelity We asked a Nuprl-familiar expert in formal logic to determine, given the Nuprl proofs and output texts, whether the texts preserved the main ideas of the formal proofs without introducing ambiguities. All 20 of our system's proofs were judged correct, while only 17 of the traditional system's proofs were judged to be correct.

5 Related Work

Nuprl creates proofs at a higher level of abstraction than other provers do, so we were able to learn verbalizations directly from the Nuprl proofs themselves. In other natural-language proof generation systems (Huang and Fiedler, 1997; Siekmann et al., 1999) and other generation applications, the semantic expressions to be realized are the product of the system's content planning component, not the proof or data. But our techniques can still be incorporated into such systems, because we can map verbalizations to the content planner's output. Hence, we believe our approach generalizes to other settings.

Previous research on statistical generation has addressed different problems. Some systems learn from verbalizations annotated with semantic concepts (Ratnaparkhi, 2000; Oh and Rudnicky, 2000); in contrast, we use un-annotated corpora. Other work focuses on *surface realization* — choosing among different lexical and syntactic options supplied by the lexical chooser and sentence planner — rather than on creating the mapping dictionary; although such work also uses lattices as input to the stochastic realizer, the lattices themselves are constructed by traditional knowledge-based means (Langkilde and Knight, 1998; Bangalore and Rambow, 2000). An exciting direction for future research is to apply these statistical surface realization methods to the lattices our method produces.

Word lattices are commonly used in speech recognition to represent different transcription hypotheses. Mangu et al. (2000) compress these lattices into *confusion networks* with structure reminiscent of our "sausage graphs", utilizing alignment criteria based on word identity and external information such as phonetic similarity.

Using alignment for grammar and lexicon induction has been an active area of research, both in monolingual settings (van Zaanen, 2000) and in machine translation (MT) (Brown et al., 1993; Melamed, 2000; Och and Ney, 2000) — interestingly, statistical MT techniques have been used to derive lexico-semantic mappings in the "reverse" direction of language *understanding* rather than generation (Papineni et al., 1997; Macherey et al., 2001). In a preliminary study, applying IBM-style alignment models in a black-box manner (i.e., without modification) to our setting did not yield promising results (Chong, 2002). On the other hand, MT systems can often model *crossing* alignment situations; these are rare in our data, but we hope to account for them in future work.

While recent proposals for *evaluation* of MT systems have involved multi-parallel corpora (Thompson and Brew, 1996; Papineni et al., 2002), statistical MT *algorithms* typically only use one-parallel data. Simard's (1999) *trilingual* (rather than multi-parallel) corpus method, which also computes MSAs, is a notable exception, but he reports mixed experimental results. In contrast, we have shown that through application of a novel composition of alignment steps, we can leverage multi-parallel corpora to create high-quality mapping dictionaries supporting effective text generation.

Acknowledgments

We thank Stuart Allen, Eli Barzilay, Stephen Chong, Michael Collins, Bob Constable, Jon Kleinberg, John Lafferty, Kathy McKeown, Dan Melamed, Golan Yona, the Columbia NLP group, and the anonymous reviewers for many helpful comments. Thanks also to the Cornell Nuprl and Columbia NLP groups, Hubie Chen, and Juanita Heyerman for participating in our evaluation, and the Nuprl group for generating verbalizations. We are grateful to Amanda Holland-Minkley for help running the comparison experiments. Portions of this work were done while the first author was visiting Cornell University. This paper is based upon work supported in part by the National Science Foundation under ITR/IM grant IIS-0081334 and a Louis Morin scholarship. Any opinions, findings, and conclusions or recommendations expressed above are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proc. of COLING*.

Regina Barzilay and Kathleen McKeown. 2001. Extracting paraphrases from a parallel corpus. In *Proc. of the ACL/EACL*, pages 50–57.

Peter Brown, Stephen Della Pietra, Vincent Della Pietra, and Robert Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Daniel Chester. 1976. The translation of formal proofs into English. *Artificial Intelligence*, 7:261–278.

Stephen Chong. 2002. Word alignment of proof verbalizations using generative statistical models. Technical Report TR2002-1864, Cornell Computer Science.

R. Constable, S. Allen, H. Bromley, W. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. 1986. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall.

Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. 1998. *Biological Sequence Analysis*. Cambridge University Press.

Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. 1999. Verbalization of high-level formal proofs. In *Proc. of AAAI*, pages 277–284.

Xiaorong Huang and Armin Fiedler. 1997. Proof verbalization as an application of NLG. In *Proc. of IJCAI*.

Tim J. P. Hubbard, Arthur M. Lesk, and Anna Tramontano. 1996. Gathering them in to the fold. *Nature Structural Biology*, 3(4):313, April.

Irene Langkilde and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proc. of ACL/COLING*, pages 704–710.

Klaus Macherey, Franz Josef Och, and Hermann Ney. 2001. Natural language understanding using statistical machine translation. In *Proc. of EUROSPEECH*.

Lidia Mangu, Eric Brill, and Andreas Stolcke. 2000. Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer, Speech and Language*, 14(4):373–400.

I. Dan Melamed. 2000. Models of translational equivalence among words. *Computational Linguistics*, 26(2):221–249.

Franz Josef Och and Hermann Ney. 2000. Improved statistical alignment models. In *Proc. of the ACL*, pages 440–447.

Alice Oh and Alexander Rudnicky. 2000. Stochastic language generatation for spoken dialogue systems. In *Proc. of the ANLP/NAACL 2000 Workshop on Conversational Systems*, pages 27–32.

Kishore A. Papineni, Salim Roukos, and R. Todd Ward. 1997. Feature-based language understanding. In *Proc. of EUROSPEECH*, volume 3, pages 1435 – 1438.

Kishore A. Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proc. of the ACL*.

Adwait Ratnaparkhi. 2000. Trainable methods for surface natural language generation. In *Proc. of the NAACL*, pages 194–201.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation System*. Cambridge University Press.

Jörg H. Siekmann, Stephan M. Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Martin Pollet, and Volker Sorge. 1999. LΩUI: Lovely ΩMEGA user interface. *Formal Aspects of Computing*, 11(3).

Michel Simard. 1999. Text-translation alignment: Three languages are better than two. In *Proc. of EMNLP/VLC*, pages 2–11.

Henry S. Thompson and Chris Brew. 1996. Automatic evaluation of computer generated text: Final report on the TextEval project. http://www.cogsci.ed.ac.uk/∼chrisbr/papers/mt-eval-final/mt-eval-final.html.

Menno van Zaanen. 2000. Bootstrapping syntax and recursion using alignment-based learning. In *Proc. of ICML*, pages 1063–1070.

Lusheng Wang and Tao Jiang. 1994. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348.