Introduction to Tactical Generation with HPSG

Woodley Packard

University of Washington

March 5, 2013

Introduction

Natural Language Generation: *the task of automatically producing natural language utterances*

- *Tactical* NLG: deciding how to convey a particular meaning
- (*Strategic* NLG: deciding what meaning to convey, when, to whom)

This NLP task dichotomy can be traced at least as far back as (McKeown 1982).

Tactical NLG

How to convey a *particular meaning* ... what do we mean by a meaning?

- Fixed shape: result of a database query or a simulation
- Unpredictable shape: general semantic representation, e.g. minimal recursion semantics [Copestake et al., 2005]

Fixed shaped meanings

Example: a weather station predicts the temperature for the next week.

- “meaning” to be conveyed: values or trend of those predictions
- possible solution, templates, e.g.:
  “Temperatures are expected to <<rise or fall>>, reaching <<extreme value>> on <<day>>.”
- Easy to produce a well-formed result; hard to make it sound both natural and unrepeatitive.

Logical forms as input

$$\text{MRS}: \quad \Big(\text{TOP} = h_1, \big\{h_{15} : \text{asleep}(x_{11}) \wedge h_1 : \text{think}(x_3, h_8)$$
$$\wedge \text{the}(x_3, h_2) \wedge \text{the}(x_{11}, h_4) \wedge h_4 : \text{dog}(x_{11})$$
$$\wedge h_2 : \text{cat}(x_3)\big\}, \big\{h_8 =_q h_{15}\big\}\Big)$$

Approximately equivalent to three predicate logics:

$$\text{Option 1} : \tilde{\exists} x_3.\text{cat}(x_3) : \text{think}(x_3, \tilde{\exists} x_{11}.\text{dog}(x_{11}) : \text{asleep}(x_{11}))$$

$$\text{Option 2} : \tilde{\exists} x_3.\text{cat}(x_3) : \tilde{\exists} x_{11}.\text{dog}(x_{11}) : \text{think}(x_3, \text{asleep}(x_{11}))$$

$$\text{Option 3} : \tilde{\exists} x_{11}.\text{dog}(x_{11}) : \tilde{\exists} x_3.\text{cat}(x_3) : \text{think}(x_3, \text{asleep}(x_{11}))$$

... which all mean the same thing (I'm using $\tilde{\exists}$ to denote this somewhat slippery "the" quantifier).

Logical forms as input

Given an MRS m and a grammar g, produce:

1. All strings s where $g(s) = m$.
   The cat thought the dog was asleep.
   The cat thought that the dog was asleep.
2. What about $g(s) \rightarrow m$? Sometimes, e.g. to let the input underspecify certain pieces of information. But no new EPs.
   The cats thought that the dogs were sleeping.
3. What about $m \rightarrow g(s)$? Not good enough.
   The cat thought.

Briefly: Motivation

In real life, what are m and s?

1. Paraphrasing: m produced by parsing another string
2. Machine translation: m produced by parsing a string in another language
3. Summarization: m is a patchwork from parses of lots of sentences
4. "Deep" template-based NLG: m is mostly static, with a few parts filled in from a DB query / weather station

But how?

1. We know how to parse:
   i.e. given an input string s and a grammar g, compute:

$$m = g(s)$$

2. We want to compute: $\{s \in \Sigma^* : m \in g(s)\}$.

Idea 1: Brute Force

$R = \{\}$
for $s \in \Sigma^*$ **do**
  compute $g(s)$
  if $m \in g(s)$ **then**
    $R = R \cup \{s\}$
  end if
end for
return R

1. Problem: complexity is atrocious (infinite).
2. Limit to at most N letters; $|\Sigma|^N$ strings to parse, each taking $O(N^3)$ time.
3. With $\Sigma = [A - Za - z0 - 9.?!]$, too slow for $N > 2$ or so.
4. We could generate *Hi*, but maybe not *Bye*

Idea 1: Post mortem

Idea 1 searched lots of strings that:

1. weren't words, e.g.:
   Zqf.9f, *ooOOf11*
2. weren't grammatical, e.g.
   dinosaurs dinosaur dinosaurs dinosaurs dinosaurs
3. weren't relevant, e.g.
   Dinosaurs drink coffee. when we want *Dogs chase cats.*

Theme: wasting time on irrelevant strings.

Idea 2: Brute Force, improved

$R = \{\}$
$V = \text{relevant\_words}(m)$
for $s \in V^*$ **do**
    compute $g(s)$
    if $m \in g(s)$ **then**
        $R = R \cup \{s\}$
    end if
end for
return R

1. Still need to limit infinite search V^* to, say, N words.
2. To generate "The cat thought the dog was asleep.", minimally need $|V| = 6$ and $N = 7$ (in practice, $|V| = 13$); $6^7 = 279936$ candidate seven-word sentences to parse at 65ms each; roughly 5 hours.
3. Tractable for modest N, but not fast.

Idea 2: Sidenote on Relevant Words

How do we compute $V = \text{relevant\_words}(m)$?

1. Any given EP in m can only be produced by a small list of grammar signs;
   straightforward to retrieve all possible grammar signs that could produce any of the input EPs.
2. That's not enough; some words are syntactically required but don't show up in the logical form at all (e.g. "was" in our example).
3. Hand-written rules to trigger vacuous lexemes

Idea 2: Post mortem

Idea 2 was a lot better than idea 1, but still wasted time on:

1. ungrammatical strings, e.g.
   asleep asleep asleep asleep asleep
2. irrelevant strings, e.g.
   The dog thought the cats were dogs.
3. Phrases like "the cat" and "the dog was asleep" may be tried and needlessly *re*parsed thousands of times as common substrings of disparate hypotheses.

Idea 3: Dynamic Programming

```
R = {}, C = {}, A = {(w, FS(w)) | w ∈ relevant_words(m)}
while a = next(A) do
    if length(a) > max_length then
        continue
    end if
    for (b, r) ∈ C × rules(g) do
        if applicable(r, a, b) then
            A.add(apply(r, a, b))
        end if
        if applicable(r, b, a) then
            A.add(apply(r, b, a))
        end if
    end for
    C.add(a)
    if meaning(a) = m then
        print R
    end if
end while
```

Idea 3: Analysis

1. Only grammatical strings are considered $\rightarrow$ much faster.
2. Don't have to parse candidates; their meaning is directly available.
3. Commenting out three lines in ACE to approximate this algorithm: "The cat thought the dog was asleep." takes about 5 minutes, explores 169618 hypotheses.
4. Lots of unnecessary hypotheses are still generated, e.g.: *as though the cat asleep was thinking*
5. New idea: a phrase whose meaning is not compatible with the goal meaning cannot be a constituent in the result. [Shieber, 1988]

Idea 4: Block Some Erroneous Hypotheses

function APPLICABLE((rule, a, b)): Boolean
    if $(rule, a, b)$ is not unifiable **then**
        return FALSE
    end if
    $m' = \text{meaning}(apply(rule, a, b))$
    if m' contradicts m **then**
        return FALSE
    else
        return TRUE
    end if
end function

1. Actual implementation: augment initial hypotheses feature structures with information from m in such a way that if m' contradicts m then $(rule, a, b)$ will not be unifiable.
2. Enabling this in ACE: "The cat thought the dog was asleep." takes 90 *milli*seconds, explores 818 hypotheses!

Other Optimizations

“Do not throw paper or other litter on the paths and in the terrain.” – 14 words, 17 EPs.

1. Idea 4: 23.6 seconds, 28647 hypotheses.
2. With ambiguity packing: 1.8 seconds, 4734 hypotheses.
3. With index accessibility filtering: 0.5 seconds, 2275 hypotheses.
4. See [Carroll and Oepen, 2005] for those optimizations.
5. Modern engines (LKB, AGREE, ACE) deploy all these optimizations.
6. Generation is frequently faster than parsing!
7. <joke> Maybe we can speed up *parsing* by enumerating all MRSes and generating from them! </joke>

Bibliography

J. Carroll and S. Oepen. High efficiency realization for a wide-coverage unification grammar. *Natural Language Processing–IJCNLP 2005*, pages 165–176, 2005.

A. Copestake, D. Flickinger, C. Pollard, and I.A. Sag. Minimal recursion semantics: An introduction. *Research on Language & Computation*, 3(2):281–332, 2005.

Kathleen R McKeown. The text system for natural language generation: An overview. In *Proceedings of the 20th annual meeting on Association for Computational Linguistics*, pages 113–120. Association for Computational Linguistics, 1982.

Stuart M Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th conference on Computational linguistics-Volume 2*, pages 614–619. Association for Computational Linguistics, 1988.