Generation — what is it exactly and can we improve it?

July 2024

A TFS grammar

Grammar A grammar consists of a set of grammar rules, G, a set of lexical entries, L, and a start structure, Q.

Lexical sign A lexical sign is a pair $\langle L, S \rangle$ of a TFS L and a string list S

Valid phrase A valid phrase P is a pair $\langle F, S \rangle$ of a TFS F and a string list S:

1. P is a lexical sign, or
2. F is subsumed by some rule R and $\langle F_1, S_1 \rangle \ldots \langle F_n, S_n \rangle$ s.t. R's daughters' subsume $F_1 \ldots F_n$ and S is the ordered concatenation of $S_1 \ldots S_n$.

Sentences A string list S is a well-formed sentence if there is a valid phrase $\langle F, S \rangle$ such that the start structure Q subsumes F.

Parsing and generation

Parsing a given string, S, consists of finding all valid phrases $\langle F_1, S\rangle \ldots \langle F_n, S\rangle$ such that the start structure Q subsumes each structure F_1 to F_n.

Generating from a start structure, Q', which is equal to or subsumed by, the general start structure Q, consists of finding all valid strings, $S_1 \ldots S_n$ which correspond to valid signs, $\langle F_1, S_1\rangle \ldots \langle F_n, S_n\rangle$ such that the start structure Q' subsumes each structure F_1 to F_n.

But this doesn't really work …

Generation from what?!

We can't instantiate Q' with a semantic structure in a particular configuration — that would mean knowing the syntax in advance.

Generation can't be "purely" semantic in any framework because of the logical form equivalence problem:

- Multiple LFs are logically equivalent.
- We can't tell which LF a grammar will accept.
- LF equivalence problem is undecidable, even for FOPC

Generation: MRS approach

Two part solution:

1. Not all formal equivalences should be treated as equivalent for generation:

$$\forall x[\mathit{student}'(x) \implies \mathit{happy}'(x)]$$
$$\neg\exists x[\mathit{student}'(x) \wedge \neg \mathit{happy}'(x)] \wedge (\mathit{happy}'(k) \vee \neg \mathit{happy}'(k))$$

2. Allow for some variation via flat semantic representation:
   2.1 $[\mathrm{this}(c), \mathrm{dog}(c), \mathrm{chase}(e, c, c'), \mathrm{the}(c'), \mathrm{cat}(c')]$
   2.2 $[\mathrm{cat}(c'), \mathrm{chase}(e, c, c'), \mathrm{dog}(c), \mathrm{the}(c'), \mathrm{this}(c)]$

Lexicalist generation (cf Shake ’n Bake)

Naively:

1. From the LF, construct a bag of lexical signs with the semantic indices instantiated (i.e., coindexation fixed).
2. List the signs in all possible orders.
3. Parse each order.

- Highly independent of syntax
- Requires lexical entries to be recoverable
- Not exactly efficient . . .

Chart generation

Lexical signs are used to instantiate the chart, then generation as parsing.

id	MRS	string	drs
	Lexical edges		
1	$\mathrm{a}'(y1)$	*a/an*	
2	$\mathrm{consultant}'(y1)$	*consultant*	
3	$\mathrm{german}'(y1)$	*German*	
4	$\mathrm{every}'(x1)$	*every*	
5	$\mathrm{manager}'(x1)$	*manager*	
6	$\mathrm{interview}'(e1_{past}, x1, y1)$	*interviewed*	
	Some of the edges constructed		
12	$\mathrm{a}'(y1), \mathrm{consultant}'(y1)$	*a consultant*	(1,2)
18	$\mathrm{every}'(x1), \mathrm{manager}'(x1)$	*every manager*	(4,5)
22	$\mathrm{interview}'(e1_{past}, x1, y1), \mathrm{a}'(y1), \mathrm{consultant}'(y1)$	*interviewed a consultant*	(6,12)
24	$\mathrm{german}'(y1), \mathrm{consultant}'(y1)$	*german consultant*	(3,2)

Lexical lookup for lexicalist generation

$$\text{a}'(y), \text{consultant}'(y), \text{german}'(y),$$
$$\text{every}'(x), \text{manager}'(x), \text{interview}'(e_{past}, x, y)$$

The instantiated lexical entry for *interview* contains:
$\text{interview}'(e1, x1, y1)$ ($e1$,$x1$ and $y1$ constants)
Complications:

- Lexical rules: past form of *interview*
- Multiple lexical entries (cf lexical ambiguity).
- Multiple relations in a lexical entry, with possible overlaps. E.g., *who* — **which_rel**, **person_rel**.
- Lexical entries without relations (e.g., infinitival *to*).

Chart generation in DELPH-IN

1. algorithm enhanced to allow semantics to be contributed by rules
2. MRS input makes construction of the bag of signs fairly easy
3. some elegant approaches for added efficiency (see Carroll et al)
4. overgeneration e.g., *big red box*, ?*red big box*. so stochastic ordering constraints

Generation formally, revisited

We have to add semantics to the tuples for the signs, because TFS subsumption doesn't allow for different orders.
A sign is a pair $\langle L, S \rangle$ of a TFS L, a string list S and an MRS M.

Generating from a start structure, Q', which is equal to or subsumed by, the general start structure Q, and has MRS, M, consists of finding all valid strings, $S_1 \dots S_n$ which correspond to valid signs, $\langle F_1, S_1, M_1 \rangle \dots \langle F_n, S_n, M_n \rangle$ such that the start structure Q' subsumes each structure F_1 to F_n and $M_1 \dots M_n$ are MRS-equivalent to M.
Note: while incorporating string append into a TFS formalism is possible, implementing MRS-equivalence efficiently in a TFS framework would be tricky . . .

Generation: the central issue

Defining MRS-equivalence:

- Not logical equivalence, so what is it?
  Defined/implemented as consistency of coindexation with identical bags of EPs, but sorts on variables (e.g., tense)? subsumption of predicates? optional EPs (e.g., parg in the ERG)?
- We'd like to make it easier for developers to build input MRSs that are acceptable, but cost in efficiency of having a relaxed notion of equivalence.
- Grammar decisions: capturing semantic nuances in MRS vs flexibility in generation?

Semantic equivalence

- Semantic equivalence is a big topic!
- I personally don't believe there's a general, workable notion of semantic equivalence, but is this generation context constrained enough to think about options?
- DMRS equivalence is neater, but fundamentally has the same issues.

Generation: other issues

- Retrieval of lexical entries imposes constraints on MRS composition (e.g., no loss of relations, no merging of relation names) — formalized in the algebra (completely?). (cf tokenization in parsing)
- Practically, for larger grammars, need to constrain the lexical entries with no semantics to the ones that might be useful for a given MRS input: error prone!
- and other things?