Semantics and generation

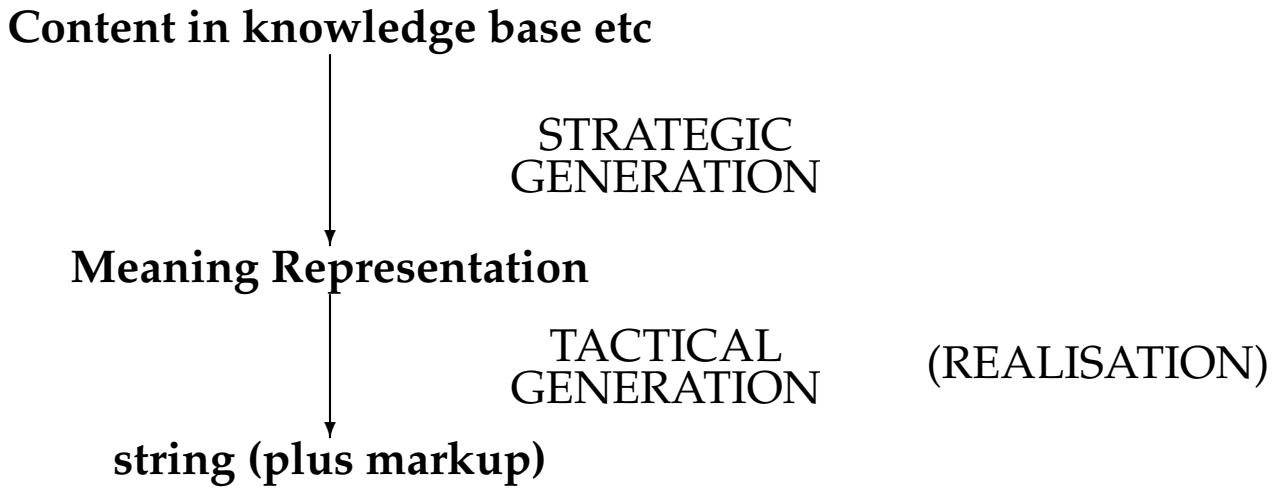
Ann Copestake

aac@cl.cam.ac.uk
Joint work with Stephan Oepen, Dan Flickinger, John Carroll

Generation from what?!

1. Meaning representation for generation
2. Chart generation and flat semantics
3. MRS and the LinGO generator
4. SEM-I
5. What next?

Some terminology



STRATEGIC GENERATION Organizing knowledge to be conveyed and constructing a meaning representation that corresponds to an utterance. In principle, grammar independent.

TACTICAL GENERATION Meaning representation to string. In principle, independent of domain knowledge.

Semantic transfer fills the same 'box' as strategic generation.

Tasks in generation

(adapted from Dale and Reiter)

Content determination deciding what information to convey.

Document structuring e.g., introduction, conclusion.

Aggregation deciding how information may be split into sentence-sized chunks.

Referring expression generation deciding when to use pronouns, what modifiers are needed, etc.

Lexical choice deciding which lexical items to use to convey a given concept.

Surface realization mapping from a meaning representation for an individual sentence to a string (or speech output).

Desiderata for a portable realization module

- Application independent
- Any well-formed input should be accepted
- No grammar-specific/conventional information should be essential in the input
- Output should be idiomatic

Semantics or quasi-semantics?

- Meaning representation must discriminate sufficently between realisations:

  Kim likes Sandy
  $like(k, s)$
  t

  Obviously not generating from truth values!

- Generation from models? Not for modular tactical generation.
- Generation from a logical form: directly or indirectly corresponding to semantics in a grammar. Generate all and only the strings with 'compatible' semantics.
  - bidirectionality
  - information structure etc

Prime examples

```
A -> B x C
B -> prime-number
C -> prime-number
```

The intended semantics is that A corresponds to the product of B and C. The semantics of B and C is stated to be the integers themselves.

Option 1: $(B'.C')$ — e.g., (2.3)
Input of 2.3 results in the string '2 x 3' but not '3 x 2'

Option 2: evaluation of B times C — e.g., 6
Input of 6 results in the string '2 x 3' and '3 x 2'
Input of a 232 digit number results in a very long wait ...
(around one year, assuming 215,000 500MHz Pentiums, each with 4Gb of RAM)

The representation quandary

- Representations must be expressive enough to correspond to desirable strings only.
- Representation must be possible to construct by strategic generation, so it shouldn't implicitly encode grammar-specific/syntactic information:

$$\begin{bmatrix} \text{pred } \textbf{and} \\ \text{arg1} \begin{bmatrix} \text{pred } \textbf{warm} \\ \text{arg1 } \boxed{0} \end{bmatrix} \\ \text{arg2} \begin{bmatrix} \text{pred } \textbf{and} \\ \text{arg1} \begin{bmatrix} \text{pred } \textbf{sunny} \\ \text{arg1 } \boxed{0} \end{bmatrix} \\ \text{arg2} \begin{bmatrix} \text{pred } \textbf{day} \\ \text{arg1 } \boxed{0} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\text{warm}(x) \wedge (\text{sunny}(x) \wedge \text{day}(x))$$

  Getting the conjunctions right would require syntax.

- BUT representation must allow efficient generation.

Logical form equivalence

Generate all and only the strings with 'compatible' semantics:

- If LF1 is generated by grammar G from string S, and LF2 is logically equivalent to LF1, then a realiser working with grammar G should accept LF2 and produce string S
- Unfortunately impossible for even first order predicate calculus (pointed out by Shieber)

Canonical representation

- Involves modification of grammar
  - prime example

    `A -> B x C`

    if $B < C$ then $(B'.C')$ else $(C'.B')$
    Input must be 2.3, but will generate '2x3' and '3x2'
  - adjective example: alphabetical order, adjectives before nouns???
- No straightforward notion of canonical representation
- Requires complex composition rules in the grammar

Is full logical equivalence what we require for 'compatibility'?

- Suppose input is:

  $$\text{not}(\text{every}(x, \text{black}(x) \wedge \text{dog}(x), \text{every}(y, \text{cat}(y), \text{likes}(x, y))))$$

  It is not the case that every black dog likes every cat.
  There's some black dog that does not like some cat.

- But classical quantifiers are very rare in applications:

  $\text{this}(x, \text{path}(x) \wedge \text{steep}(x) \wedge \text{slippery}(x), \text{the}(y, \text{stream}(y), \text{follow}(x, y))))$

  This steep slippery path follows the stream.
  This slippery steep path follows the stream.
  The stream is followed by this steep slippery path. ...

- Intuition about the most strict notion of compatibility: same predications, equivalence of 'grammatical' conjunction. Broaden this by underspecification, not inference.

- Semantics or quasi-semantics?

What is required?

1. A realisation algorithm that is efficient with minimal guidance from the logical form syntax.
2. A formalism that allows sufficient abstraction over syntax that constructing the input to the realiser is doable.
3. An interface layer expressing those constraints imposed by the grammar that have to be known to construct the input.
4. A fluency component that ranks realisations.

The LinGO approach

1. Realisation algorithm: chart generation.
2. Formalism: MRS (and RMRS)
   - flatness
   - algebra that guarantees composition properties
   - underspecifiability
3. Interface layer: SEM-I (in progress).
4. Fluency: different statistical models.

Naive lexicalist generation

1. From the LF, construct a bag of instantiated lexical signs.
2. List the signs in all possible orders.
3. Parse each order.

- Highly independent of syntax
- Requires lexical entries to be efficiently recoverable directly from LF (easiest with flat semantics)
- Not exactly efficient ...
- Shake and Bake generation was part of an approach to MT in which transfer operates across instantiated lexical signs (Whitelock et al)
- Shake and Bake isn't as bad as the totally naive approach, but still worst-case exponential

Lexical lookup for lexicalist generation

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

(ignoring scope for the moment)

1. For each elementary predication, find the corresponding lexical sign (i.e., TFS for HPSG)
2. Set the argument positions in the lexical sign to constant values - e.g., interview'(c2,c1,c3), manager'(c1), consultant'(c3), german'(c3), a'(c3), every'(c1)
   This means that unification ensures that the predicate-argument structure is correct.

Chart generation (Kay, 1996 — and others)

Lexical signs are used to instantiate the chart.

Generation as chart parsing:

Lexical edges

index	sem	cat	string
x	manager(x)	N	*manager*
x	the(x)	Det	*the*
e	work(e,x)	VP	*worked*

Edges constructed

x	manager(x), the(x)	NP	*the manager*
e	work(e,x), the(x), manager(x)	S	*the manager worked*

Chart generation, continued

1. indexing of the chart can be done by semantic indices
2. no overlap: check as the edges are constructed that semantics is only used once
3. completeness check at the end
4. some restrictions on the grammar:
   - daughters may not overlap — e.g., cannot have semantics constructed by means of multiple inheritance between types contributed from two sources
   - monotonicity: none of the components may be removed when constructing a phrase
5. still worst case exponential (because of adjectives etc)

- MRS involves lists of elementary predications (EPs).
- Mostly, one EP corresponds to one lexical sign — 'surfacy' representation (attempt to capture syntax and productive morphology, but not lexical semantics).
- Every EP has a label. Scope is represented by relationship between 'hole' and label: qeq (equality modulo quantifiers).

$$l1 : \mathsf{a}(y, h2, h3), l4 : \mathsf{consultant}(y), l4 : \mathsf{german}(y),$$
$$l6 : \mathsf{every}(x, h7, h8), l9 : \mathsf{manager}(x), l10 : \mathsf{interview}(e, x, y)$$
$$h2 =_q l4, h7 =_q l9$$

- Lexical lookup corresponds to finding lexical entries that correspond to EP (indexed by predicate).
- Algebra which guarantees non-overlap and monotonicity: EPs are accumulated by append.

Complications for the lookup phase

- Hierarchy of relations: e.g., **_open_v** subsumes **_open_v_1**, **_open_v_2**, etc
- Multiple lexical entries (cf lexical ambiguity).
- Multiple EPs in a lexical entry: e.g., *who* — **which_q_rel**, **person_rel**
  Note possible overlaps: *where* — **which_q_rel**, **place_rel**
- Lexical entries without relations (null semantics): e.g., infinitival *to*.
  Currently, hand-written rules triggered by MRS are required to license introduction.
- EPs contributed by constructions: e.g., prpstn_m_rel
- EPs contributed by lexical rules.

Null semantics introduction rule

```
there_expl_rule := generator_rule &
 [ CONTEXT [ RELS <! [ LBL handle,
                       PRED _be_v_there_rel ] !> ],
   OUTPUT "there_expl" ].


which_rel_rule := generator_rule &
 [ CONTEXT [ RELS <! [ LBL #hand,
                       ARG0 ref-ind ],
                     [ LBL #hand,
                       PRED prpstn_m_rel ] !> ],
   OUTPUT "which_r" ].
```

Chart generation in the LKB (July 2004 version!)

1. constant values contained in the INSTLOC feature
2. quick check vector (this makes indexing chart by semantic indices redundant)
3. packing
4. accessibility conditions
5. matching qeqs option
6. stochastic ordering constraints

Packing in generation (Oepen and Carroll)

- Packing in parsing: combining edges which are compatible. In constraint-based grammar: exclude parts of semantics that don't restrict (list of EPs, qeqs), combine edges when there's subsumption. e.g., PPs.
- Packing in generation: same thing! Works because:
  - indices are in valence features etc, so they still constrain unification
  - overlap conditions can be checked without looking at the feature structure, because strict accumulation of EPs

Accessibility conditions

- Generating with intersective modifiers:

  $l1 : \mathsf{a}(y, h2, h3), l4 : \mathsf{consultant}'(y), l4 : \mathsf{german}'(y),$
  $l6 : \mathsf{every}'(x, h7, h8), l9 : \mathsf{manager}'(x), l10 : \mathsf{interview}'(e, x, y)$

  Want to avoid partial structure:

  every manager interviewed a consultant

- Two phase generation (Carroll et al, 1999). Intersective modifiers added by adjunction — but this requires conditions on the intersective modifier rules.
- Accessibility: check in non-semantic part of the feature structure to make sure indices still needed are available.

qeqs

- ordinary index equality is guaranteed by INSTLOC constants
- qeqs look as though they need a separate check
- BUT, pretend that qeqs are equalities and equate INSTLOCs between qeq HARG and LARG in grammar and in input to generator
- if indices were actually equated, these structures wouldn't scope, but this doesn't matter if it's only INSTLOCs

Constructing the input MRS

- In effect, assumes that lexical choice for open class words and most closed class words has been done.
- Underspecification possible (but reduces efficiency, although packing helps).
- Also requires knowledge of 'grammar predicates': i.e., predicates that don't correspond to lexical items but are introduced by morphology, constructions and some closed class words (where there's decomposition).
- SEM-I (semantic interface) is intended as the interface definition (API in programming terms).

- Object-level SEM-I. Contains information for the EPs corresponding to words in the language (especially open-class). Automatically constructed from the lexicon. For example:

lexeme	string	relation	ARG0	ARG1	ARG2	eg	doc
act_v1	act	_act_v_rel	event	oblig	index	oblig	

- Meta-level SEM-I. A semi-manually-constructed fully-documented database of all the relations introduced via constructions and via lexical types, and of all the values which may appear on semantic features that occur in the (R)MRSs. For example:

relation	ARG0		ARG1		ARG2		doc	eg
appos_rel	event	oblig	index	oblig	index	oblig	< link >	< eg num >

Guaranteed generation?

Given well-formed input, with all elementary predications found in SEM-I, can we generate a string?

- Semantically bleached lexical items: some uses of *which*, *one*, *piece*, *make* etc are just about making the syntax 'work'. For instance:

$$\text{every}(x, \text{dog}(x) \wedge \text{sleep}(e_{past}, x), \text{bark}(e'_{pres}, x))$$

  every dog which slept is barking
  every dog which was asleep is barking

- Defective paradigms, negative polarity, anti-collocations etc?
- with input fix up?
- negotiation?

Conclusions

- Semantics for generation is about form as much as content.
- Meaning representation is an interface layer, hence need to consider the issues in constructing it as well as using it.
- Goal of realisation cannot be stated in terms of logical equivalence.
- MRS is an attempt to find a good balance between expressivity and efficiency.
- In general, we make use of general MRS algebra properties for efficiency.
- Efficiency is now reasonable, but we need to make the input MRS easier to construct.
- SEM-I should provide stability/documentation.

Next steps for semantics and generation in the LKB

- Further work on the SEM-I
- Generation from RMRS
- Hierarchies outside the grammar
- Lexical semantic classes
- Semi-automatic techniques for creating 'null semantic' rules
- Relax requirements for input MRS:
  - allow MRSs with more underspecification
  - underspecified quantifiers, integrating determiner prediction component
  - 'fix up' of MRSs that won't generate because of missing grammatical predicates