

Semantics and generation

Ann Copestake

aac@cl.cam.ac.uk

Natural language generation utilising linguistically-motivated, general-purpose, grammars can conveniently be thought of as involving two components. The first component involves constructing some logical form that is accepted by the grammar, while the second (**tactical generation**, or now more usually, **realisation**) involves going from the logical form to a string. What I want to discuss here is mainly realisation, and in particular, the implications that supporting tractable realisation has for the nature of logical form in computational grammars. The title of this talk is perhaps a little misleading, because what I am concerned with is not denotation, but with the form of the semantic representation and with the way that composition operates.

To start off with, I'll consider an artificial example. Suppose we have a grammar consisting of the following rules:

```
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. Now consider two options for representing the semantics associated with the rule $A \rightarrow B C$:

1. The semantics for A is an expression representing the product, say $B'.C'$ (e.g., 2.3).
2. The semantics for A is the evaluated expression: i.e., the integer that results from the multiplication (e.g., 6).

Suppose we choose the second option, and try producing a realisation from the input 6. The grammar would output "2 x 3" and "3 x 2". But suppose instead that the input is a 232 digit number which is the product of two large primes, n and m . Generating from this will be rather slow ...¹ Of course, if we chose instead to represent the product as in the first option (i.e., as the expression $n.m$) it is trivial in all cases to obtain the realisation corresponding to "n x m", because we do not have to perform the factorisation. However, as things stand, we would not get the output "m x n".

There are two inter-related issues here that are relevant to natural language generation:

1. We can make the realisation problem more or less difficult, depending on the meaning representation we choose, even though the denotation is the same. That is, the difficulty depends on the syntax of the semantics. Of course, we could make the realisation problem trivial by making the semantic representation very close to natural language (e.g., claiming that the semantics of 'the dog chased the cat' was 'the dog chased the cat') but it should be obvious that we cannot go too far in this direction. The question of whether the generation problem as a whole is doable depends on whether the input to the realiser can be constructed. The assumption I am going to make here is that the realiser is application-independent. For some applications, a relatively 'surfacy' meaning representation is all that is required, machine translation by semantic transfer being a good example. In other cases, the application itself may utilise meaning representations that are considerably further away from natural language. But if we do more work in the grammar to make the logical forms closer to the application, we lose application-independence and make realisation more difficult.
2. Structure in the logical form may be exploited to guide realisation, and this may be necessary for realisation to work at all. But structure that reflects the syntax of the language too closely is generally to be avoided, since it can make it impossible for the input to the realiser to be constructed without detailed knowledge of the grammar. To elaborate the example above slightly: suppose that the grammar also has a linear precedence constraint, such that it only generates strings of the form "B x C" where B is numerically less than C. In this case, input to the realiser of 3.2 will fail to generate: the system constructing the input needs to know about the linear precedence constraint. Of course, this particular example is trivial, but for a natural language, the wrong choice of semantic representation could result in the input to the realiser requiring very fine-grained knowledge of the grammar, such as whether a particular predication was syntactically realised as a pre- or post- modifier.

¹Possibly around one year, assuming 215,000 500MHz Pentiums, each with 4Gb of RAM: see <http://www.rsasecurity.com/rsalabs/>.

Alternatively, suppose that there is no linear precedence constraint, but the semantic construction rule is stated so that the rule $A \rightarrow B C$ has an associated semantic composition rule that says that the semantics is $B'.C'$ if $B' < C'$ and is $C'.B'$ otherwise (so a canonical order is imposed). In this case, 2.3 should generate both “2 x 3” and “3 x 2”. 3.2 will fail to produce a realisation, but if the canonical order is part of the specification of the semantic representation language, this is reasonable. Versions of canonical representation have, in fact, been proposed as a solution to the syntactic dependence problem (e.g., Landsbergen, 1987). But the idea has serious disadvantages: semantic composition and the generation algorithm are both seriously complicated by the need to support canonicity.

A better solution is to make the semantic representation less dependent on syntax. So, for our trivial example, we can allow the realiser to permute the factors, and express the product in a way that makes this clear: e.g. $\{2, 3\}$. The generation algorithm is required to treat $\{2, 3\}$ as equivalent to $\{3, 2\}$. There's no particular advantage to the alternative notation for this trivial example, since we could have simply required the generator to treat 2.3 and 3.2 as the same, but the alternative notation can be extended to other operators.

Some of these issues have been discussed in terms of the problem of ‘logical form equivalence’ in realisation (e.g., Shieber, 1993). Shieber points out that there is no possibility of defining a generator that can accept an arbitrary first-order logical expression: even if a logically-equivalent form is acceptable, the equivalence problem is not decidable. Even if we weaken the logic, so that the equivalence problem is decidable, we won't necessarily have a tractable realisation algorithm. So the goal of generating from any form logically equivalent to one directly accepted by the grammar is not feasible. I am not aware of any work that has precisely specified an alternative goal for robust realisation, but from a practical perspective what is needed is:

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.

In the talk, I will discuss the extent to which we've made progress with these goals, and what remains to be done.

References

- Landsbergen, J. (1987) ‘Isomorphic Grammars and their use in the ROSETTA Translation System’ in M. King (ed.), *Machine Translation Today: The State of the Art*, Edinburgh University Press, pp. 351–372.
- Stuart Shieber (1993) ‘The problem of logical form equivalence’, *Computational Linguistics*, **19** (1), 179–190.