Ambiguity

Last time we talked about Montague’s 1973 method of obtaining multiple representations for a statement to reflect the ambiguity. This essentially involved replacing a statement like “Every girl pets a dog” by “A dog, every girl pets it” to obtain the second meaning.

The idea is that in computing the meaning of “every girl pets it” we must replace it by a variable, z, and then lambda-abstracting it. That is, the meaning is:

$$\lambda z. \forall x. (\text{girl}(x) \rightarrow \text{pets}(x, z))$$

Because the meaning of “a dog” is $$\lambda v. \exists y. (\text{dog}(y) \land v @ y)$$, the meaning of the entire statement is:

$$(\lambda v. \exists y. (\text{dog}(y) \land v @ y)) (\lambda z. \forall x. (\text{girl}(x) \rightarrow \text{pets}(x, z)))$$

We decided that this was unsatisfactory because it required a rephrasing of the sentence in order to get the appropriate semantics, and that seemed like a cheat. Another problem is that often we can’t disambiguate a sentence until later:

“Every girl pets a dog. There were so many girls that they scared it.” vs. “Every girl pets a dog. All the dogs were wagging their tails.”

Cooper Storage

Cooper 1975 (revised in 1983) presented a different technique that did not require rewriting parse trees, though did not yet meet all of our goals. It is essentially a minor variant of Montague’s techniques that is essentially a notational variant of the parse tree, that allows one to postpone making a decision about the order of quantifiers as long as desired.

We illustrate the technique with an example:

$$\langle \text{PET}(z_1, z_2),$$
$$\langle \lambda u. \forall x. (\text{girl}(x) \rightarrow u @ x), 1 \rangle$$
$$\langle \lambda u. \exists y. (\text{dog}(y) \land u @ y), 2 \rangle \rangle$$

Every girl pets a dog

\[\text{Every girl} \quad \text{pets a dog} \]

$$\langle \lambda w. (w @ z_1),$$
$$\lambda u. \forall x. (\text{girl}(x) \rightarrow u @ x), 1 \rangle \rangle$$

$$\langle \lambda u. \text{PET}(u, z_2),$$
$$\lambda u. \exists y. (\text{dog}(y) \land u @ y), 2 \rangle \rangle$$

The idea is that whenever you encounter a noun phrase \(\phi\), you may replace it by \(\langle \lambda u. (u @ z_i), \phi \rangle\). We say that \(\phi\) is frozen and placed in storage.
Definition 1 If the store \( \langle \phi, (\beta, j), \ldots , (\beta', k) \rangle \) is a semantic representation for a quantified NP, then the store \( \langle \lambda u.(u@z_1), (\phi, i), (\beta, j), \ldots , (\beta', k) \rangle \), where \( i \) is some unique index, is also a representation for that NP.

Once we have built this representation, we can complete the semantics by taking things out of storage in any order desired.

Definition 2 Retrieval: Let \( \sigma_1 \) and \( \sigma_2 \) be (possibly empty) sequences of binding operators. If the store \( \langle \phi, \sigma_1, (\beta, i), \sigma_2 \rangle \) is associated with an expression of category \( S \), then the store \( \langle \beta@\lambda z_1.\phi, \sigma_1, \sigma_2 \rangle \) is also associated with this expression.

Going back to our example, we pull the second element out of the store first and get:

\[
\langle \lambda u.(\forall x.(girl(x) \rightarrow u@x))(\lambda u.\exists y.(dog(y) \land u@y), 2) \rangle \\
= \langle \forall x.(girl(x) \rightarrow (\lambda z_1.\forall x.(girl(x) \rightarrow PET(x, z_1)))@x), (\lambda u.\exists y.(dog(y) \land u@y), 2) \rangle \\
= \langle \forall x.(girl(x) \rightarrow PET(x, z_2)), (\lambda u.\exists y.(dog(y) \land u@y), 2) \rangle 
\]

Then we can pull out the last element and get:

\[
\langle (\lambda u.\exists y.(dog(y) \land u@y))(\lambda z_2.\forall x.(girl(x) \rightarrow PET(x, z_2))) \rangle \\
= \exists y.(dog(y) \land (\lambda z_2.\forall x.(girl(x) \rightarrow PET(x, z_2)))@y) \\
= \exists y.(dog(y) \land \forall x.(girl(x) \rightarrow PET(x, y))) 
\]

We can implement this in a computer program. Of course we’d like to get both meanings, so that we can decide between them, though recall that we could also have left either or both in place, so we need to take those into consideration. However, many of those are redundant, so we could write a program that later filters out alphabetic variants.