An Informal but Respectable Approach to Computational Semantics – Part II

CS224N/Ling 237 – Christopher Manning

7 Database/knowledgebase interfaces

We will assume that the predicates of our logical representations are available as tables in a database. We'll assume a straightforward mapping of one table per predicate. In practice, there would often be more indirect mappings (for instance, a unary **red** predicate could come from a table that lists objects and their colors), but we'll assume we have any necessary *views* set up to make the mapping straightforward. Let's do the easiest example again:

S : **respect**(**fong**)(**kathy**)

NP : **kathy** VP : **respect**(**fong**)

Kathy V : **respect** NP : **fong**

respects *Fong*

Here we assume that **respect** is a table Respect with two fields respecter and respected, and that **kathy** and **fong** are IDs in the database: k and f. If this was a statement, we would evaluate the form **respect**(**fong**)(**kathy**) by doing an insert operation:

insert into Respects(respecter, respected) values (k, f)

Below, we will focus on questions like the corresponding *Does Kathy respect Fong* for which we will use the relation to ask:

select 'yes' from Respects where Respects.respecter = k and Respects.respect = f

(we interpret "no rows returned" as **no**). We would now like to know how to work with more complex examples. I'll show some (hopefully correct!) SQL, but we will only work through the translation from English to logical representations. The translation to SQL is then "in principle" straightforward, since relational databases implement a simple form of logic (cf. Datalog).

8 Nominal Type Shifting

We have a reasonable semantics for *red car in Palo Alto* as a property from **Ind** → **Bool**, but what about when we want to represent noun phrases like *the red car in Palo Alto* or *every red car in Palo Alto*? In general, we will need to use methods of *type-shifting* for nominal expressions. The diagram below shows common patterns of nominal type shifting:

$$\begin{tikzcd} \mathbf{Ind} \arrow[rr, shift left, "\mathbf{Q}"] \arrow[dr, "\mathbf{R}"'] & & \mathbf{Ind}\to\mathbf{Bool} \arrow[ll, dashed, shift left, "\iota"] \arrow[dl, "\mathbf{some}^2"] \\ & (\mathbf{Ind}\to\mathbf{Bool})\to\mathbf{Bool} & \end{tikzcd}$$

$\mathbf{R}(\mathrm{x}) = \lambda P.P(x)$
$\mathbf{some}^2(\mathrm{P}) = \lambda Q.(Q \cap P) \neq \emptyset$
$\mathbf{Q}(\mathrm{x}) = \lambda y.x = y$

Let's start with *the* to which we gave without explanation the translation ι in the first handout. $[\![\iota]\!](P) = a$ if $(P(b) = \textbf{yes}$ iff $b = a)$. We use this as a semantics for *the* following Russell, for whom *the x* meant the unique item satisfying a certain description (in reality things get a little more complex, I might note). ι is thus only a partial function, returning undefined if there is no such object (undefined represents a presupposition failure which is technically different from a false statement).

Let's continue the example from last time doing now *the red car in Palo Alto*:

$$\text{NP} : \iota(\lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x))$$

$$\text{Det} : \iota \qquad \text{N}' : \lambda x.\mathbf{car}(x) \wedge \mathbf{in'}(\mathbf{paloalto})(x) \wedge \mathbf{red'}(x)$$

$$\textit{the} \qquad \textit{red car in Palo Alto}$$

For *red car in Palo Alto* we have the property semantics from before, which we render in SQL as:

```
select Cars.obj from Cars, Locations, Red where Cars.obj = Locations.obj AND
Locations.place = 'paloalto' AND Cars.obj = Red.obj
```

(here we assume the unary relations have one field, obj). The effect of ι would then be rendered via a having clause:

```
select Cars.obj from Cars, Locations, Red where Cars.obj = Locations.obj AND
Locations.place = 'paloalto' AND Cars.obj = Red.obj
having count(*) = 1
```

9 Generalized Quantifiers

What then of *every red car in Palo Alto*? If we want to translate a sentence like *Every red car in Palo Alto is expensive*, then we need to make a quantificational statement that involves *red cars in Palo Alto* and *expensive things*. A generalized determiner is a relation between two properties, one contributed by the restriction from the N′, and one contributed by the predicate quantified over:

$$(\mathbf{Ind} \to \mathbf{Bool}) \to (\mathbf{Ind} \to \mathbf{Bool}) \to \mathbf{Bool}$$

(In other literature, these are generally called "generalized quantifiers", but we take quantifiers to take a single property argument, and to map from $(\mathbf{Ind} \to \mathbf{Bool}) \to \mathbf{Bool}$, that is to be things of the type at the bottom of the type-shifting triangle. A generalized determiner maps from a property to a quantifier.) Here are some other determiners:

$\mathbf{some}^2(\mathbf{kid})(\mathbf{run}) \equiv \mathbf{some}(\lambda x.\mathbf{kid}(x) \wedge \mathbf{run}(x))$
$\mathbf{every}^2(\mathbf{kid})(\mathbf{run}) \equiv \mathbf{every}(\lambda x.\mathbf{kid}(x) \to \mathbf{run}(x))$
Exercise: $\mathbf{no}^2(\mathbf{kid})(\mathbf{run}) \equiv$
$\mathbf{two}^2(\mathbf{kid})(\mathbf{run}) \equiv$

Which are implemented via the quantifiers:

$\mathbf{every}(P) = \textbf{yes}$ iff $(\forall x)P(x) = \textbf{yes}$; i.e., if $P = \mathbf{Dom_{Ind}}$
$\mathbf{some}(P) = \textbf{yes}$ iff $(\exists x)P(x) = \textbf{yes}$; i.e., if $P \neq \emptyset$

The central insight of Montague's PTQ (proper theory of quantification) was that the same trick could be applied to indviduals, representing them as quantifiers (which are type-raised individuals):

Kathy : $\lambda P.P(\mathbf{kathy})$

The syntactic category of noun phrases is thus realized uniformly in the semantic dimension. (This was both good, and bad – everything was always raised to the most complicated type needed for anything of the category. We'd prefer to allow more flexible type shifting.)

In the type-shifting diagram above, **R** is exactly this basic type-raising function for individuals. Later, we also need nominal type-raising and lowering within functional types as follows (Hendriks 1993, simplified):

Value raising raises a function that produces an individual as a result to one that produces a quantifer. If $\alpha : \sigma \to \mathbf{Ind}$ then $\lambda x.\lambda P.P(\alpha(x)) : \sigma \to (\mathbf{Ind} \to \mathbf{Bool}) \to \mathbf{Bool}$

Argument raising replaces an argument of a boolean function with a variable and applies the quantifier semantically binding the replacing variable. If $\alpha : \sigma \to \mathbf{Ind} \to \tau \to \mathbf{Bool}$ then $\lambda x_1.\lambda Q.\lambda x_3.Q(\lambda x_2.\alpha(x_1)(x_2)(x_3)) : \sigma \to (\mathbf{Ind} \to \mathbf{Bool}) \to \mathbf{Bool} \to \tau \to \mathbf{Bool}$

Argument lowering replaces a quantifier in a boolean function with an individual argument, where the semantics is calculated by applying the original function to the type raised argument. If $\alpha : \sigma \to ((\mathbf{Ind} \to \mathbf{Bool}) \to \mathbf{Bool}) \to \tau \to \mathbf{Bool}$ then $\lambda x_1.\lambda x_2.\lambda x_3.\alpha(x_1)(\lambda P.P(x_2))(x_3) : \sigma \to \mathbf{Ind} \to \tau \to \mathbf{Bool}$

10 Questions with answers!

Questions will not actually have a special logical representation. A yes/no question (*Is Kathy running?*) will be something of type **Bool**, but if it is a question we will interrogate our database to see if it is true or not, rather than either asserting or checking the fact as for a statement. We could put something like a question mark before it to indicate this interpretation, but I won't. A content question (*Who likes Kathy?*) will be an *open proposition*, that is something semantically of the type *property* (**Ind** $\to$ **Bool**), and operationally we will consult the database to see what individuals will make the statement true. For the kind of semantic forms we produce, it is fairly straightforward to evaluate them in a model or against a database or a knowledge base.

We will need new syntax rules for questions, new lexical items, and empty missing NPs, which include a means of *gap threading* to link them up with their filler, the *wh*-phrase. I won't give formal rules, but empty gapped elements are carried up the tree, until the variable is reintroduced at some (sentential) node. (This idea of gap threading is formalized in various syntactic theories like GPSG/HPSG.) Under our analysis, auxiliaries make no contribution to meaning – that's reasonable enough as what meaning contribution they do make is in areas like tense, which we aren't dealing with.

S' : $\beta(\alpha) \to$ NP[*wh*] : β Aux S : α	*who*, NP[*wh*] : $\lambda U.\lambda x.U(x) \wedge \mathbf{human}(x)$
S' : $\alpha \to$ Aux S : α	*what*, NP[*wh*] : $\lambda U.U$
NP/NP$_z$: $z \to$ e	*which*, Det[*wh*] : $\lambda P.\lambda V.\lambda x.P(x) \wedge V(x)$
S : $\lambda z.F(\ldots z \ldots) \to$ S/NP$_z$: $F(\ldots z \ldots)$	*how_many*, Det[*wh*] : $\lambda P.\lambda V.\lvert\lambda x.P(x) \wedge V(x)\rvert$

- $\text{S'} : \lambda z.\mathbf{like}(z)(\mathbf{kathy})$
  - $\text{NP}[wh] : \lambda U.U$
    - *What*
  - Aux
    - *does*
  - $\text{S} : \lambda z.\mathbf{like}(z)(\mathbf{kathy})$
    - $\text{S/NP}_z : \mathbf{like}(z)(\mathbf{kathy})$
      - $\text{NP} : \mathbf{kathy}$
        - *Kathy*
      - $\text{VP/NP}_z : \mathbf{like}(z)$
        - $\text{V} : \mathbf{like}$
          - *like*
        - $\text{NP/NP}_z : z$
          - *e*

select liked from Likes where Likes.liker='Kathy'

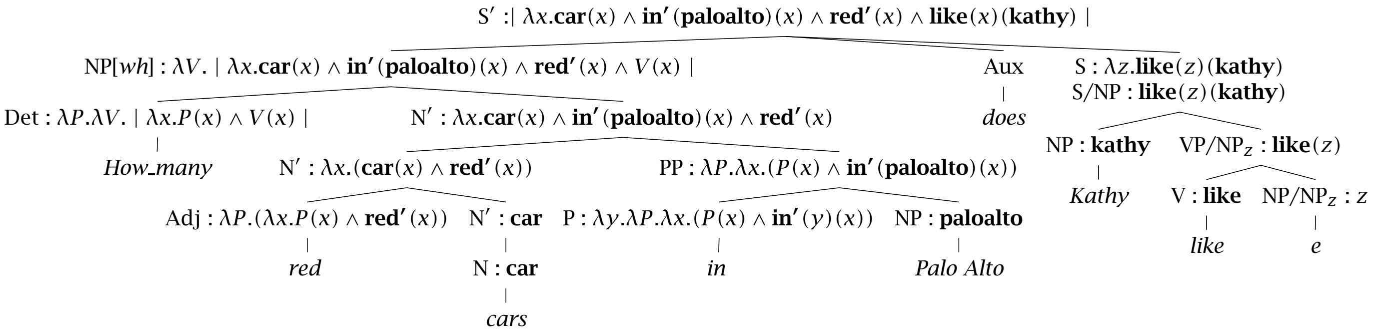
- $\text{S'} : \lambda x.\mathbf{like}(x)(\mathbf{kathy}) \wedge \mathbf{human}(x)$
  - $\text{NP}[wh] : \lambda U.\lambda x.U(x) \wedge \mathbf{human}(x)$
    - *Who*
  - Aux
    - *does*
  - $\text{S} : \lambda z.\mathbf{like}(z)(\mathbf{kathy})$
    - $\text{S/NP}_z : \mathbf{like}(z)(\mathbf{kathy})$
      - $\text{NP} : \mathbf{kathy}$
        - *Kathy*
      - $\text{VP/NP}_z : \mathbf{like}(z)$
        - $\text{V} : \mathbf{like}$
          - *like*
        - $\text{NP/NP}_z : z$
          - *e*

select liked from Likes,Humans where Likes.liker='Kathy' AND Humans.obj=Likes.liked

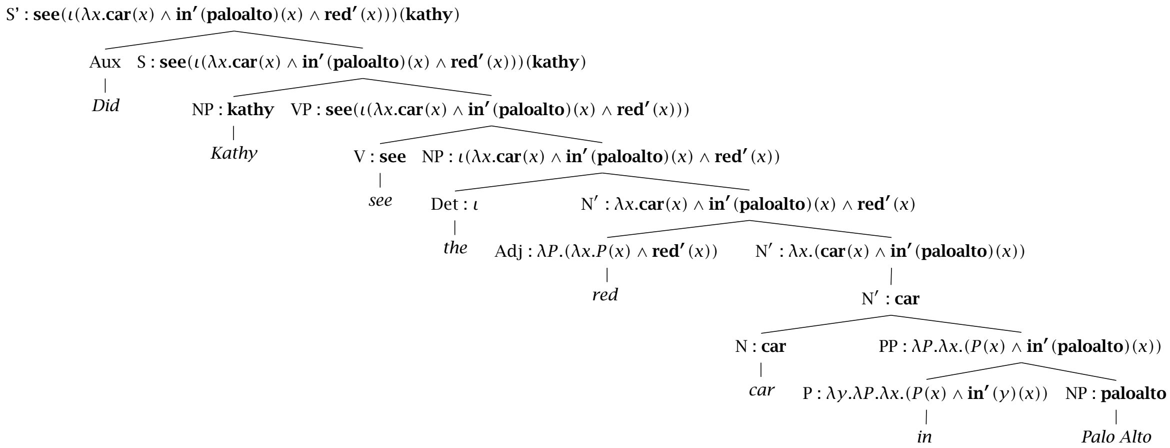
- $\text{S}' : \lambda x.\mathbf{car}(x) \wedge \mathbf{like}(x)(\mathbf{kathy})$
  - $\text{NP}[wh] : \lambda V.\lambda x.\mathbf{car}(x) \wedge V(x)$
    - $\text{Det} : \lambda P.\lambda V.\lambda x.P(x) \wedge V(x)$
      - *Which*
    - $\text{N}' : \mathbf{car}$
      - $\text{N} : \mathbf{car}$
        - *cars*
  - Aux
    - *did*
  - $\text{S} : \lambda z.\mathbf{like}(z)(\mathbf{kathy})$
    - $\text{S/NP} : \mathbf{like}(z)(\mathbf{kathy})$
      - $\text{NP} : \mathbf{kathy}$
        - *Kathy*
      - $\text{VP/NP}_z : \mathbf{like}(z)$
        - $\text{V} : \mathbf{like}$
          - *like*
        - $\text{NP/NP}_z : z$
          - *e*

select liked from Cars,Likes where Cars.obj=Likes.liked AND Likes.liker='Kathy'

- $\text{S}' : \lambda x.\mathbf{car}(x) \wedge \mathbf{every}^2(\mathbf{student})(\mathbf{like}(x))$
  - $\text{NP}[wh] : \lambda V.\lambda x.\mathbf{car}(x) \wedge V(x)$
    - $\text{Det} : \lambda P.\lambda V.\lambda x.P(x) \wedge V(x)$
      - *Which*
    - $\text{N}' : \mathbf{car}$
      - $\text{N} : \mathbf{car}$
        - *cars*
  - Aux
    - *did*
  - $\text{S} : \lambda z.\mathbf{every}^2(\mathbf{student})(\mathbf{like}(z))$
    - $\text{S/NP} : \mathbf{every}^2(\mathbf{student})(\mathbf{like}(z))$
      - $\text{NP} : \mathbf{every}^2(\mathbf{student})$
        - $\text{Det} : \mathbf{every}^2$
          - *every*
        - $\text{N}' : \mathbf{student}$
          - *student*
      - VP
        - $\text{NP}_z : \mathbf{like}(z)$
          - $\text{V} : \mathbf{like}$
            - *like*
          - NP
            - $\text{NP}_z : z$
              - *e*



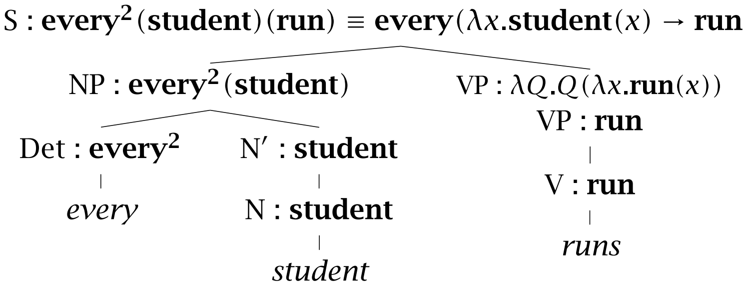
select count(*) from Likes,Cars,Locations,Reds where Cars.obj = Likes.liked AND Likes.liker = 'Kathy' AND Red.obj = Likes.liked AND Locations.place = 'Palo Alto' AND Locations.obj = Likes.liked



select 'yes' where Seeings.seer = k AND
Seeings.seen = (select Cars.obj from Cars, Locations, Red where Cars.obj =
Locations.obj AND Locations.place = 'paloalto' AND Cars.obj = Red.obj
having count(*) = 1)

11 Noun phrase scope

We'll first do *Every student runs*. We could just redefine things to allow the subject to take the verb phrase as its argument semantically, but we can maintain the usual idea of what is the functor and what is the argument by alternatively doing **argument raising** on the VP.



Following Hendriks (1993), we can use nominal type shifting – argument raising and lowering – to do (a fair amount of) the analysis of scopal ambiguities, for example, the two readings of *Some kid broke every toy* – one where one particular kid broke them all, and the other where different kids broke different toys:

- S : $\mathbf{every}^2(\mathbf{toy})(\lambda y_o.\mathbf{some}^2(\mathbf{kid})(\lambda x_s.\mathbf{break}(y_o)(x_s)))$
  - NP : $\mathbf{some}^2(\mathbf{kid})$
    - Det : $\mathbf{some}^2$
      - *some*
    - N′ : $\mathbf{kid}$
      - N : $\mathbf{kid}$
        - *kid*
  - VP : $\lambda S'.\mathbf{every}^2(\mathbf{toy})(\lambda y_o.S'(\lambda x_s.\mathbf{break}(y_o)(x_s)))$
    - V : $\lambda O.\lambda S'.O(\lambda y_o.S'(\lambda x_s.\mathbf{break}(y_o)(x_s)))$
      V : $\lambda x_o.\lambda S.S(\lambda x_s.\mathbf{break}(x_o)(x_s))$
      V : $\lambda y.\lambda x.\mathbf{break}(y)(x)$
      - *broke*
    - NP : $\mathbf{every}^2(\mathbf{toy})$
      - Det : $\mathbf{every}^2$
        - *every*
      - N′ : $\mathbf{toy}$
        - N : $\mathbf{toy}$
          - *toy*

- S : $\mathbf{some}^2(\mathbf{kid})(\lambda y_s.\mathbf{every}^2(\mathbf{toy})(\lambda x_o.\mathbf{break}(x_o)(y_s)))$
  - NP : $\mathbf{some}^2(\mathbf{kid})$
    - Det : $\mathbf{some}^2$
      - *some*
    - N′ : $\mathbf{kid}$
      - N : $\mathbf{kid}$
        - *kid*
  - VP : $\lambda S.S(\lambda y_s.\mathbf{every}^2(\mathbf{toy})(\lambda x_o.\mathbf{break}(x_o)(y_s)))$
    - V : $\lambda O'.\lambda S.S(\lambda y_s.O(\lambda x_o.\mathbf{break}(x_o)(y_s)))$
      V : $\lambda x_s.O(\lambda x_o.\mathbf{break}(x_o)(x_s))$
      V : $\lambda y.\lambda x.\mathbf{break}(y)(x)$
      - *broke*
    - NP : $\mathbf{every}^2(\mathbf{toy})$
      - Det : $\mathbf{every}^2$
        - *every*
      - N′ : $\mathbf{toy}$
        - N : $\mathbf{toy}$
          - *toy*

References

It's harder than you might think to find good things to read on computational semantics for NLP. Jurafsky & Martin present some of the basic implements for doing computational semantics, but it is hard to take seriously a fragment that can't even handle a sentence like *Who did you call?* [and they just do translation to markerese]. Russell and Norvig also avoid dealing with quantifiers. The presentation here draws most closely from Bob Carpenter, *Type-Logical Semantics* (MIT Press 1999), but reformulates things so that I don't have to introduce categorial grammar but use rule-to-rule translations more in the tradition of Bach (1976)/GPSG/HPSG. (Many semanticists like categorial grammar because of the nice connections it provides between the syntactic and semantic theories. In particular, things we have specified, like type-raising rules, are theorems of the Lambek calculus.) There are other intros to semantics from a more linguistic perspective, including De Swart, *Introduction to Natural Language Semantics* (CSLI) and Gamut, *Logic, Language and Meaning* (University of Chicago Press, 1991), which are sound, but don't mention computational issues. There's also Heim and Kratzer, *Semantics in Generative Grammar* (Blackwell, 1997), which I haven't seen but from the table of contents it looks like it might be heavy on Chomskyan syntax. Finally, Patrick Blackburn and Johan Bos, *Representation and Inference for Natural Language* http://www.comsem.org/ provide a comprehensive treatment of computational semantics, implementing everything in Prolog.