Matthew Francis-Landau
69%
Semantics AS.050.317
midterm
2016-11-28

1. Gricean reasoning:

For B's utterance, it is implicating that he has some cat but that it is not a siamese cat.

In the B' utterance where B says no, B is being truthful in that he is communicating that he does not have a "siamese cat," (x such that $\textsc{siamese}(x) \wedge \textsc{cat}(x)$) however, it does not communicate which of these statements is false. By saying "I have a cat", B is asserting that $\textsc{cat}(x) = 1$, but that $\textsc{siamese}(x) \wedge \textsc{cat}(x) = 0$ which more specifically specifies which part of the statement is false.

This follows Grice's cooperative principals as it concisely communicates what is false rather then just saying "no" and leaving two possible interpretations.

This is a good explanation of why B' is mor informative than B 15/15
but it doesnt discuss why the implicature arises; which of Grice's submaxims is being flouted or violated? 0/15
Alos should do a cancellation test to show that this is an implicature and not an entailment 0/3

2. Possessive and more:

(See attached IPython notebook)

3. Expressive adjectives:

(a) information contributed to sentence

The adjective is modifying the noun *computer* to contain the fact that the speaker views the computer unfavorably. Good, but is it always negative? Give some data to show how the context could push this around. What happens if you move ADJ around in the sentnece? Do all the words listed (darn, damn, fucking...) work exactly the same? 5/11

$$\{x|\textsc{computer}(x) \wedge \textsc{darn}(x)\} \subseteq \{x|\textsc{computer}(x)\}$$

The context here might be relevant in determining exactly what DARN means in that some speakers might say "darn" before everything, while another might use it more sparingly and thus the fact it is use communicates a lot about the speakers internal belief.

(b) What type is the contribution.

This statement is conversational implicature as we can see that it isn't an at-issue entailment or presupposition since we could imagine that this statement could be canceled if the speaker said something like "*Actually, this is a wonderful computer.*" Further, we can take "darn"

Good
But it also projects out of contexts that presuppositions do...and its not clear what Gricean reasoning could derive the implicature here. We're just not dealing with the literal meaning of fucking...

8/11

as meaning that the speaker does not like the computer and thus by attaching additional meaning to "computer" through an adjective.

We can see a similar structure in "*Look at those stupid people over there.*" Here we have that "*people over there*" is of the same type as *computer*, and we have attached the adjective *stupid* similar to *darn*. It is clear in this context, that *stupid* classification remains attached to these people after this statement has been made and does not require that we continue reference them as *stupid*. (A *darned computer* stays *darned* even if we start just calling it a *computer*.)

(c) What is the type

$$ADJ_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda f_{\langle e,t\rangle}.\lambda x_e.(ADJ(x) \wedge f(x))$$

Good

With adjectives we have to be able to combine a number of these expressions. (If we say the "blue darned computer" then we want the same type as "darned computer") We already have that "computer" represents a set of objects by having a function which will restrict entities to being of a computer type, and we have adjectives as taking this class and adding further restrictions in that the object is also "darned" or "fucked".

Good

11/11

The exact meaning of "darned" or "fucked" etc will have to depend on the context and the entity that is being described.

Good

(d) Differences in meaning across expressive adjectives

In the case of listed adjectives, we might try and place these adjectives on a single scale from really "bad"/"unpleasant" to "wonderful" ($\textsc{damn}(x) = \textsc{enjoyment}(\textsc{relativeDarnScore}_c(x), x)$).

As I have done earlier in the problem, we can have a different function which represents each of these items. This is a very "abstracted" representation of these adjectives as it does not require supposing that there is some single scale or representation that can be used with these words.

9/11

But ideally we would like to have a single function for all ADJs
The difference could be encoded in the degree operator you have suggested. The other dimensio you need to address is the fact that different ADJs seems to have different intensities (e.g. Darn is not as intense as fucking)

2. Possessives

Here is a basic lexicon for most of sentences like (1-3):

```
(1) Alfonso talked to Joanna's mother.
(2) Alfonso borrowed Joanna's book.
(3) Alfonso kicked Joanna's chair.
```

The only parts that I haven't defined are the possessive morpheme, and `mother`.

In [2]:
```
%%lamb
||Alfonso|| = a_e
||Joanna|| = j_e
||book|| = L x_e : Book_<e,t>(x)
||chair|| = L x_e : Chair_<e,t>(x)
||borrowed|| = L x_e : L y_e : Borrow(y,x)
||kick|| = L x_e : L y_e : Kick(y,x)
||talk to|| = L x_e : L y_e : Talkto(y,x)
```

```
INFO (meta): Coerced guessed type t for 'Borrow_t' into <(e,e),t>, to match a
rgument '(y_e, x_e)'
INFO (meta): Coerced guessed type t for 'Kick_t' into <(e,e),t>, to match arg
ument '(y_e, x_e)'
INFO (meta): Coerced guessed type t for 'Talkto_t' into <(e,e),t>, to match a
rgument '(y_e, x_e)'
INFO (parsing): Exporting item ||talk to|| to python variable `talk_to`.
```

Out[2]:
$$[\![\textbf{Alfonso}]\!]_{e} = a_{e}$$
$$[\![\textbf{Joanna}]\!]_{e} = j_{e}$$
$$[\![\textbf{book}]\!]_{\langle e,t\rangle} = \lambda x_{e} \,.\, Book(x_{e})$$
$$[\![\textbf{chair}]\!]_{\langle e,t\rangle} = \lambda x_{e} \,.\, Chair(x_{e})$$
$$[\![\textbf{borrowed}]\!]_{\langle e,\langle e,t\rangle\rangle} = \lambda x_{e} \,.\, \lambda y_{e} \,.\, Borrow(y_{e}, x_{e})$$
$$[\![\textbf{kick}]\!]_{\langle e,\langle e,t\rangle\rangle} = \lambda x_{e} \,.\, \lambda y_{e} \,.\, Kick(y_{e}, x_{e})$$
$$[\![\textbf{talk\_to}]\!]_{\langle e,\langle e,t\rangle\rangle} = \lambda x_{e} \,.\, \lambda y_{e} \,.\, Talkto(y_{e}, x_{e})$$

The following cell defines an entry that you can use for `the`. Since there are no presuppositions per se in the lambda notebook, this glosses over them, but you can think of the `Iota` operator as itself being presuppositional.

In [3]:
```
%%lamb
c_{e} = c_{e}
||the|| = L f_<e,t> : Iota x_e : (f(x) & x in c_{e})
```

Out[3]:
$$c_{\{e\}} = c_{\{e\}}$$
$$[\![\textbf{the}]\!]_{\langle\langle e,t\rangle,e\rangle} = \lambda f_{\langle e,t\rangle} \,.\, \iota x_{e} \,.\, (f_{\langle e,t\rangle}(x_{e}) \wedge (x_{e} \in c_{\{e\}}))$$

Part [a]: In the following blank cell, calculate using * the truth-conditions of:

```
(4) Alfonso borrowed the book.
```

In [17]:
```
# composition here
Alfonso * (borrowed * (the * book))
```

Out[17]:
1 composition path. Result:

[0]: $[\![[[\textbf{borrowed [the book]}]\ \textbf{Alfonso}]]\!]_{t} = Borrow(a_{e}, \iota x_{e} \,.\, (Book(x_{e}) \wedge (x_{e} \in c_{\{e\}})))$

Here $x_e \in c_{\{e\}}$ from the "*the*" restrict the set of books to which ever item is in the context and is associated with the statement "*the*".

Part [b]: In the following cell, write an entry for `mother`, and describe it in the next cell.

In [73]:
```
%%lamb
llmotherll = λ x_e : Mother(x)
```

```
INFO (meta): Coerced guessed type t for 'Mother_t' into <e,t>, to match argument 'x_e'
```

Out[73]: $[\![\textbf{mother}]\!]_{\langle e,t\rangle} = \lambda x_e \,.\, Mother(x_e)$

Markdown cell: please explain your entry for `mother` in prose by editing this cell.

The type of mother is $\langle e, t\rangle$ which is similar to "*cat*" as we can write "*Alfonso's cat*" and we do not want "*cat*" to be only a isngle e entity type.

Part [c] (possessive denotation): In the following cell write an entry for the possessive morpheme. To handle python naming, call it `POSS` instead of `'s`.

In [72]:
```
%%lamb
llPOSSll = λ f_<e,t> : λ o_e : Ιota m_e : (f(m) & PossRelation(m, o))
```

```
INFO (meta): Coerced guessed type t for 'PossRelation_t' into <e,e),t>, to match argument '(m_e, o_e)'
```

Out[72]: $[\![\textbf{POSS}]\!]_{\langle\langle e,t\rangle,\langle e,e\rangle\rangle} = \lambda f_{\langle e,t\rangle} \,.\, \lambda o_e \,.\, \imath m_e \,.\, (f_{\langle e,t\rangle}(m_e) \wedge PossRelation(m_e, o_e))$

Markdown cell: please explain your entry for `POSS` in prose by editing this cell.

(Note: $PossRelation(\cdot,\cdot)$ might be better written as $(m, o) \in c_{\text{poss relations}}$ where c represents context and information about the relation between the m entity and the o (owner) entity.)

The type for the possessive is $\langle\langle e, t\rangle, \langle e, e\rangle\rangle$ as we are going to take a set of objects (such as "cat" or "mother") of type $\langle e, t\rangle$ and some grounded entity (such as "Alfonso" or "Joanna") of type e perform a $PossRelation$ between these objects which may depend on the context/background information.

Part [c] (explication of why this denotation works): In the following cell, calculate the meaning of the tree in (6) on the midterm using the * operator.

In [67]:
```
Joanna * (POSS * mother)
```

Out[67]: 1 composition path. Result:

[0]: $[\![\textbf{[[POSS mother] Joanna]}]\!]_e = \imath m_e \,.\, (Mother(m_e) \wedge PossRelation(m_e, j_e))$

Comments: The type of this expression is e which is what we want. We can see this if we consider some context such as "*Joanna's mother is Lilly*," then we want "*Lilly*" and "*Joanna's mother*" to work interchangeably and "*Lilly*" clearly has the type e.

In [29]:
```
Alfonso * talk_to * (Joanna * (POSS * mother))
```

Out[29]: 1 composition path. Result:

[0]:

$[\![\textbf{[[talk_to Alfonso] [[POSS mother] Joanna]]}]\!]_t = Talkto(\imath m_e \,.\, (Mother(m_e) \wedge PossRelatio$

Part [d] (grads, ugrad extra credit). In the following cell(s) answer the question on the midterm. You may insert as many new cells as you need, and use markdown cells to write any prose.

LF = Logical forms

Given the structure on possessives, we are taking "*name 's mother*" to be an e type which is dependent on the context and requires us to interpert this statement. We could have a process which simplifies this expression by performing the lookup into the context (eg if we had in our context "*Joanna's mother is Lilly*" then we can replace "*Joanna's mother*" with "*Lilly*" and the meaning of the statement doesn't change).

Another alternate would be to generate some abstract entity which we associate with "*Joanna's mother*." This could work by creating some E_{456} when we encounter "Joanna's mother" and then replacing this statement with E_{456} and adding to our context "Joanna's mother is E_{456}"

In [ ]:

Part [d] In the following cells, calculate the denotation of `A doctor met Joanna's mother`. You will need to fill out the lexicon with a few more items, and then do the composition.

In [45]:
```
%%lamb # rest of lexicon here
||doctor|| = L d_e : Doctor(e)
||a|| = L f_<e,t> : Iota x_e : (f(x) & x << c)
```

```
INFO (meta): Coerced guessed type t for 'Doctor_t' into <e,t>, to match argum
ent 'e_e'
INFO (meta): Coerced guessed type t for 'Met_t' into <(e,e),t>, to match argu
ment '(x_e, y_e)'

Don't know what to do with '# rest of lexicon here'
```

Out[45]:
$$[\![\textbf{doctor}]\!]_{\langle e,t\rangle} = \lambda d_e \,.\, Doctor(e_e)$$
$$[\![\textbf{a}]\!]_{\langle\langle e,t\rangle,e\rangle} = \lambda f_{\langle e,t\rangle} \,.\, \iota x_e \,.\, (f_{\langle e,t\rangle}(x_e) \wedge (x_e \in c_{\{e\}}))$$
$$[\![\textbf{met}]\!]_{\langle e,\langle e,t\rangle\rangle} = \lambda x_e \,.\, \lambda y_e \,.\, Met(x_e, y_e)$$

In [21]:
```
# composition here
```

In [46]:
```
(a * doctor) * met * (Joanna * (POSS * mother))
```

Out[46]: 1 composition path. Result:

[0]:

$$[\![[\textbf{[met [a doctor]] [[POSS mother] Joanna]}]\!]_t = Met(\iota x_e \,.\, (Doctor(e_e) \wedge (x_e \in c_{\{e\}})), \iota m_e \,.$$

Comments: Here "*a*" is selecting a specific entity from the set of doctors and then we have the same as before with the mother which then means that "*met*" just converts some relationship between two entities into a truth condition.

In [ ]:

3. Expressive adjectives

If you like, you can do some or all of this problem in the lambda notebook; I won't provide a template though.

In [65]:
```
%%lamb
||broke|| = L o_e : L p_e : Broke(p, o)
||computer|| = L c_e : Computer(c)
||darn|| = L a_<e,t> : L b_e : (Darn(b) & a(b))
```

```
INFO (meta): Coerced guessed type t for 'Broke_t' into <(e,e),t>, to match ar
gument '(p_e, o_e)'
INFO (meta): Coerced guessed type t for 'Computer_t' into <e,t>, to match arg
ument 'c_e'
INFO (meta): Coerced guessed type t for 'Darn_t' into <e,t>, to match argumen
t 'b_e'
```

Out[65]:

$[\![\textbf{broke}]\!]_{\langle e,\langle e,t\rangle\rangle} = \lambda o_e \,.\, \lambda p_e \,.\, Broke(p_e, o_e)$
$[\![\textbf{computer}]\!]_{\langle e,t\rangle} = \lambda c_e \,.\, Computer(c_e)$
$[\![\textbf{darn}]\!]_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle} = \lambda a_{\langle e,t\rangle} \,.\, \lambda b_e \,.\, (Darn(b_e) \wedge a_{\langle e,t\rangle}(b_e))$

In [66]:
```
Alfonso * (broke * (the * (darn * computer)))
```

Out[66]:

1 composition path. Result:

[0]:

$[\![[[\textbf{broke [the [darn computer]]] Alfonso}]\!]_t = Broke(a_e, \iota x_e \,.\, ((Darn(x_e) \wedge Computer(x_e)) /$

In [ ]:
```
```