# Dynamic semantics

Simon Charlow (simon.charlow@rutgers.edu)

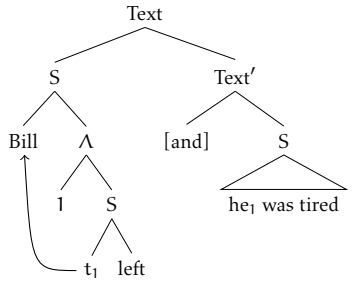
December 2 & 7, 2015

## 1 Discourse referents

- Today, we'll explore the following idea: discourse referent ('dref') introduction means assignment function modification.
- More concretely: by the time we get around to processing the second sentence of (1), the assignment function we use has changed: it maps 9 to the senator who admires Kennedy.

  (1) [Exactly one senator]$_9$ admires Kennedy. He$_9$'s very junior.

- Of course, assignment shifting is *already* a part of our theory. In the LF below, $t_1$ ends up evaluated at an assignment that maps 1 to Bill, i.e., $g[1 \to \text{B}]$.




  However: PA doesn't allow *he*$_1$ to get evaluated at this shifted assignment too. Assignment shifting is tied too closely to the presence of a c-commanding (at LF!) abstraction index.

- Today we liberate dref introduction from LF c-command. (Along the way, we'll arrive at an arguably more compositional treatment of variables than our official theory managed.)
- You can think of this as one way of making precise what H&K are doing in Chapter 11 (in particular, giving us a handle on *about-ness*), with the additional benefit that it will allow a uniform account of E-type and garden-variety (i.e. free and bound) pronouns.

## 2 Meanings as assignment shifters

- Our departure point: discourse referent introduction is assignment function modification: the meaning of a sentence is itself the sort of thing that can update an assignment function.

- Sentence meanings map assignment functions into *new* assignment functions. Type: $\langle \mathsf{a}, \mathsf{a} \rangle$.
- So e.g. the meaning of *Bill*$_n$ *left* is a device for taking an assignment and updating it:

$$[\![\text{Bill}_n \text{ left}]\!] = \lambda g.\, g[n \to \textsc{b}]$$

- Another way to write this, more iconically, following Heim (1983):

$$g + [\![\text{Bill}_n \text{ left}]\!] = g[n \to \textsc{b}]$$

# 3 Formally

- This is the basic picture: meaning as **context change potential** (where we are working with a somewhat denuded notion of what a context is).[1]
- But what happened to truth conditions? We need a way to distinguish *true* sentences from false ones. But the functional $\langle \mathsf{a}, \mathsf{a} \rangle$ perspective doesn't allow this. The only options are apparently success or undefinedness.[2]
- Solution: sentence meanings are functions from **discourse states** into new discourse states, where states are modeled as *sets* of assignments. I.e., their type is $\langle \langle \mathsf{a}, \mathsf{t} \rangle, \langle \mathsf{a}, \mathsf{t} \rangle \rangle$.
- E.g., here's a state with 3 assignments; it exhibits **certainty** about the middle and last discourse referent (resp. $\textsc{c}$ and $\textsc{e}$), and **uncertainty** about the remaining three discourse referents.

$$\left\{ \begin{array}{l} \textsc{a}\,\textsc{a}\,\textsc{c}\,\textsc{d}\,\textsc{e}, \\ \textsc{b}\,\textsc{b}\,\textsc{c}\,\textsc{d}\,\textsc{e}, \\ \textsc{a}\,\textsc{c}\,\textsc{c}\,\textsc{e}\,\textsc{e} \end{array} \right\}$$

- Here is a representative example.

$$[\![\text{Bill}_n \text{ left}]\!] = \lambda G.\, \{g[n \to \textsc{b}] : g \in G \wedge \textsc{left}\,\textsc{b}\}$$

- More iconically:

$$G + [\![\text{Bill}_n \text{ left}]\!] = \{g[n \to \textsc{b}] : g \in G \wedge \textsc{left}\,\textsc{b}\}$$

- If Bill *didn't* leave, the result will be $\varnothing$! Truth and falsity thus correspond, respectively, to whether you return a non-empty output or an empty one!
- Sentences with pronouns filter out assignments in the input state that don't assign the pronoun to (e.g.) someone who was tired. In general, pronouns *reduce* uncertainty in the discourse.

$$G + [\![\text{he}_n \text{ was tired}]\!] = \{g : g \in G \wedge \textsc{tired}\,(g\,n)\}$$

- So, in sum, we've enriched the perspective a little bit. Instead of just talking about truth relative to an assignment, we have a notion of meaning that allows us to talk about truth relative to an assignment, as well as assignment change.

[1] Philosophers: is the notion of context at play here the same as Kaplan's? Different? Related?

[2] There are ways around this difficulty, in fact, but they have not been explored very much in the linguistics literature. If you're interested, I take this up in the second chapter of my dissertation.

# 4 Dynamic conjunction

- If sentence meanings are devices for updating assignment functions, the job of *and* must be to compose two updates—that is, to create an assignment function pipeline from the input to the first conjunct to the second conjunct to the output!

$$[\![S_1 \text{ and } S_2]\!] = \lambda G.\, [\![S_2]\!]([\![S_1]\!]\, G)$$

- And in the more iconic notation:

$$G + [\![S_1 \text{ and } S_2]\!] = (G + [\![S_1]\!]) + [\![S_2]\!]$$

- $S_1$ shifts the context of evaluation for $S_2$. In principle, then, drefs introduced by $S_1$ are available for the evaluation of $S_2$. Incorporates a notion of *order*! Conjunction is no longer commutative: the $S_1$-modified assignments are fed to $S_2$, and not vice versa.
- Here's what this gives for *Bill$_2$ left; he$_2$ was tired*:

$$\begin{aligned}
&(G + [\![\text{Bill}_2 \text{ left}]\!]) + [\![\text{he}_2 \text{ was tired}]\!] \\
&= \{g[2{\to}\textsc{b}] : g \in G \wedge \textsc{left}\,\textsc{b}\} + [\![\text{he}_2 \text{ was tired}]\!] \\
&= \left\{h : h \in \{g[2{\to}\textsc{b}] : g \in G \wedge \textsc{left}\,\textsc{b}\} \wedge \textsc{tired}\,(h\,2)\right\} \\
&= \{g[2{\to}\textsc{b}] : g \in G \wedge \textsc{left}\,\textsc{b} \wedge \textsc{tired}\,(g[2{\to}\textsc{b}]\,2)\} \\
&= \{g[2{\to}\textsc{b}] : g \in G \wedge \textsc{left}\,\textsc{b} \wedge \textsc{tired}\,\textsc{b}\}
\end{aligned}$$

- Most importantly, $G + [\![\text{Bill}_2 \text{ left}]\!]$ gives a new discourse state, one where all the assignments map 2 to Bill! These are then fed to the pronoun, which thus necessarily returns Bill as well!
- Notice that if either Bill left or wasn't tired, $\varnothing$ is returned. The discourse gets short-circuited.

# 5 Lexical semantics

## 5.1 Verbs

- Abbreviate the type of a dynamic discourse update, i.e. $\langle\langle \mathsf{a}, \mathsf{t}\rangle, \langle \mathsf{a}, \mathsf{t}\rangle\rangle$, as $\pi$.
- Intransitives like *left* or *was tired* can be modeled as functions from an individual into an update, i.e. with type $\langle \mathsf{e}, \pi\rangle$:

$$[\![\text{left}]\!] = \lambda x.\, \lambda G.\, \{g : g \in G \wedge \textsc{left}\,x\}$$

- Transitives like *likes* and *licked* can be modeled as functions from two individuals into an update, i.e. with type $\langle \mathsf{e}, \langle \mathsf{e}, \pi\rangle\rangle$:

$$[\![\text{saw}]\!] = \lambda x.\, \lambda y.\, \lambda G.\, \{g : g \in G \wedge \textsc{saw}\,x\,y\}$$

- In both cases, what's returned is simply the unchanged input discourse state if the standard truth-condition is met. Otherwise, nothing is returned, and we have observed a **failure**.

### 5.2 DPs

- Proper names denote simple individuals, type $e$!

$$[\![\text{Bill}]\!] = \textsc{b}$$

- Here's an example:

$$\begin{array}{lll} [\![\text{Bill left}]\!] = [\![\text{left}]\!]\,[\![\text{Bill}]\!] & & \textsf{FA} \\ \quad = \left(\lambda x.\, \lambda G.\, \{g : g \in G \wedge \textsc{left}\, x\}\right) \textsc{b} & & \text{Lexicon} \times 2 \\ \quad = \lambda G.\, \{g : g \in G \wedge \textsc{left}\, \textsc{b}\} & & \beta \end{array}$$

- No drefs yet?? They're coming.
- Pronouns are more complicated. They take a dynamic property $\kappa$ and an input state $G$, find out what the pronoun means at the points in the input state, and pass that individual to $\kappa$.

$$[\![\text{he}_n]\!] = \lambda\kappa.\, \lambda G.\, \left\{h : g \in G \wedge h \in \{g\} + \kappa\,(g\,n)\right\}$$

- Pronouns have a type reminiscent of quantificational DPs, i.e. $\langle\langle e, \pi\rangle, \pi\rangle$. (Why not type $e$?)
- Don't worry too much about the semantics of pronouns. It's rather annoyingly complicated, but the result will be as expected:

$$[\![\text{he}_n \text{ was tired}]\!] = \lambda G.\, \{g : g \in G \wedge \textsc{tired}\,(g\,n)\}$$

## 6 Dref introduction

- We can actually, finally, define a compositional version of the abstraction index. Its key function is that it modifies the input state $G$, then evaluates $p$ with respect to this updated state.

$$[\![n]\!] = \lambda p.\, \lambda x.\, \lambda G.\, \{h : h \in G[n{\to}x] + p\}$$

- This relies on an obvious generalization/abuse of the assignment modification notation:

$$G[n{\to}x] \coloneqq \{g[n{\to}x] : g \in G\}$$

- So an asbtraction index combines with a dynamic proposition $p$, and returns a dynamic property! It does this by rewriting all the assignments in the input state $G$!
- An example (exercise: calculate this yourself!):

$$[\![n\ [\text{t}_n \text{ left}]]\!] = \lambda x.\, \lambda G.\, \{g[n{\to}x] : g \in G \wedge \textsc{left}\, x\}$$

- Passing in *Bill*:

$$[\![\text{Bill}\ [n\ [\text{t}_n \text{ left}]]]\!] = \lambda G.\, \{g[n{\to}\textsc{b}] : g \in G \wedge \textsc{left}\, \textsc{b}\}$$

# 7 Adding in indefinites

- An issue with indefinites: *which* discourse referent gets introduced by e.g. *a linguist*? As it happens, the dynamic semantics doesn't require us to make a choice!
- That is, indefinites may introduce drefs just like proper names, but they may do so in a way that *introduces uncertainty* into the discourse state.
- Nothing about our basic setup needs to change to accommodate this possibility. We've already assumed that sentences output sets of assignments, so everything will type out!
- First, we begin with a basic semantics for an indefinite. There is no update yet. The indefinite simply works by feeding the linguists and G one-by-one to a scope argument $\kappa$ to get a bunch of new contexts, and then returns all those possibilities:

$$[\![\text{a linguist}]\!] = \lambda\kappa.\,\lambda G.\,\{h : \textsc{ling}\,x \wedge h \in G + \kappa\,x\}$$

- Dref introduction happens via the abstractor $n$. The LFs are thus *exactly* as they were in the static semantics. But the difference is that dref introduction (assignment modification) persists outside the c-command domain of $n$.

$$[\![\text{a linguist}\,[n\,[t_n\,\text{left}]]]\!] = \lambda G.\,\{g[n{\to}x] : g \in G \wedge \textsc{ling}\,x \wedge \textsc{left}\,x\}$$

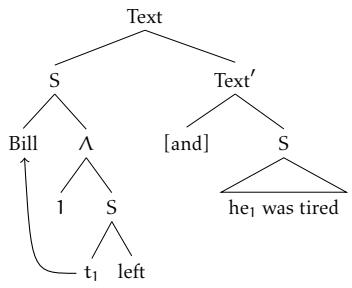
- Compare *Bill left* (repeated):

$$[\![\text{Bill}\,[n\,[t_n\,\text{left}]]]\!] = \lambda G.\,\{g[n{\to}\textsc{b}] : g \in G \wedge \textsc{left}\,\textsc{b}\}$$

- Both the indefinite and the proper name update the assignments in G. The only difference: the indefinite does so *nondeterministically*, potentially creating uncertainty.

# 8 Binding?

- In what sense does dynamic binding count as binding?
- The key diagnostic, I would suggest, is *covariation*. Consider the following LF again:




- In the static semantics, *Bill* and *he*$_1$ **accidentally corefer** — a single assignment g is passed to both the first and second sentences in the text, and g 1 happens to be $\textsc{b}$. Thus, if you changed *Bill* to, say, *John*, it would have no effect on the interpretation of *he*$_1$.

- By contrast, in dynamic semantics, there is some (not-so-spooky) action at a distance: wiggling *Bill* will cause the interpretation of $he_1$ to change accordingly.

# 9 Dynamically closed things

- Consider the data in (2). Assume (for simplicity) that the structure of the first sentence is (3).

  (2) I don't own a $\text{car}_i$. *$\text{It}_i$'s a Hyundai.

  (3) [not [$_\Sigma$ I own a car]]

- As we've seen, $\Sigma$ introduces a discourse referent. This suggests that the role of negation is to, in Karttunen terms, *delete* any discourse referents introduced by its complement $\Sigma$. Here's one way to do this:

$$G + [\![\text{not } S]\!] = \{g : g \in G \wedge \{g\} + [\![S]\!] = \varnothing\}$$

- In other words, negation simply *filters* the input context, in particular only allowing pass assignments in G that make S false. Most importantly for present purposes, any drefs generated in S are simply ignored.
- Exercise: what are the implications of this sort of theory of negation for cases like (4)? How about cases like (5) and (6)? How about (7) (this one's especially advanced, and requires you to see how the choice-functional account of exceptional scope plays with the dynamic account of exceptional anaphora — would make a great term paper topic!)?

  (4) I don't like $\text{Bill}_i$. $\text{He}_i$'s boring.

  (5) I don't like any man ⟨who criticizes $\text{Barack}_i$⟩. $\text{He}_i$'s a good president.

  (6) It isn't the case that I don't own a $\text{radio}_i$. $\text{It}_i$'s a Panasonic.

  (7) If ⟨a rich relative of $\text{mine}_i$ is feeling generous⟩, I'll get a car for my birthday.
  …$\text{She}_i$'s a steel magnate.

- A variety of other meanings are taken to be dynamically closed—that is, built on negation:

  (8) If $\text{someone}_i$ knocked, $\text{she}_i$ left. *I saw $\text{her}_i$.

# 10 Quantifiers

- So we have a pretty good handle on how both proper names and indefinites can bind outside their scope. But what about anaphora to quantifiers like *exactly one*?

  (9) Exactly one $\text{senator}_i$ admires Kennedy. $\text{She}_i$'s very junior.

- We'd like a binding analysis of the pronoun in this text, but we do *not* want the pronoun to be bound in the same way as, e.g., the trace of *exactly one senator*.

- How to do this? Here is a meaning for *exactly one senator* that accomplishes this:

$$\begin{aligned}[\![\text{exactly one senator}_n]\!] = \lambda\kappa.\,\lambda G.\,\{g[n \to \iota x.\, x \in S] : g \in G\} \\ \text{if } |S| = 1 \\ \text{otherwise } \varnothing \\ \text{where } S = \{x : \textsc{senator}\, x \wedge G + \kappa\, x \neq \varnothing\}\end{aligned}$$

- This looks complicated, but it's not doing anything too fancy. It checks that exactly one senator has property $\kappa$. If so, it overwrites the index $n$, putting **that senator** there. Otherwise, it fails by returning $\varnothing$. (Exercise: what happens to drefs generated by, e.g., indefinites in the scope of *exactly one linguist*? Is this a good prediction? [Again, this is pretty advanced, and would make an excellent term paper.])
- In-scope binding (e.g., of traces, and pronouns c-commanded at LF) can still happen via the abstraction operator, but it is crucially distinguished from out-of-scope binding: the latter, but not the former, involves anaphora to the only individual with such-and-such a property.
- How about donkey anaphora? Examples like (10) suggest that drefs generated in the restrictor NP of *every* are accessible in its scope.

  (10) Every farmer who owns a donkey$_i$ beats it$_i$.

- Here is a meaning for the quantified determiner *every* that accomplishes this: $[\![\text{every}]\!]$ checks that every individual $x$ **and assignment** $h$ satisfying the restrictor $P$ also satisfies the scope $\kappa$. If so, it returns the input state $G$ unchanged. If not, it fails.

$$\begin{aligned}[\![\text{every}]\!] = \lambda P.\,\lambda\kappa.\,\lambda G.\,\big\{g : g \in G \wedge \\ \forall x.\,\forall h.\, h \in G + P\, x \Rightarrow \\ \exists j.\, j \in \{h\} + \kappa\, x\big\}\end{aligned}$$

- Don't sweat the details, but do notice that the modified assignments $h$ output by the restrictor are *passed to* $\kappa$! So the restrictor $P$ determines a *local context* for the scope $\kappa$, one which harbors any drefs generated in the restrictor.
- This entry also predicts that any drefs generated in the restrictor should be inaccessible in subsequent sentences. This seems largely correct:

  (11) *…It$_i$ brayed.