Slides for week 2, 10 Feb 2021, Dyanmic semantics seminar

- available at the repo, https://github.com/cb125/Dynamics

Roberts talk this Friday, 12 Feb, 10:15am New York time:

From the abstract:

- "Context on this approach constitutes the scoreboard (Lewis 1979) of the language game in play. . .
- Each utterance is a move which functions to update the scoreboard. . .
- Semantic content is compositionally derived and static. . .
- [O]nly consideration of the CG, QUD and interlocutors' evident goals and intentions reflected in G [goals] permits one to infer the intended force of an utterance, i.e. the way in which the speaker proposes to update the context with its contextually resolved semantic content."

https://nyu.zoom.us/j/95020189746

Plan for 9 Feb 2021

1. Review Heim
2. Make Heim pointwise
3. Present DPL
4. Compare Heim w/DPL
5. Make DPL eliminative
6. Discuss order of evaluation
7. Consider how to add a pragmatic transmission

Review of Heim's 1983 fragment

- A context C is a set of pairs of assignments and worlds
- Clauses denote Context Change Potentials (CCPs): $C \rightarrow C$
- A special context '#' represents presupposition failure
- Conjunction: c + "A and B" = (c + A) + B
- Indefinites: c + "a_i P Q" = (c + P(i)) + Q(i)
  - Novelty presup: c maps i to every individual in every world
- Negation: c + "Not A" = c\(c + A)
  - Presuppositions project appropriately, but not dynamically closed

Heim's negation is not dynamically closed

```
c1 = [([g],n) | g <- [1..5], n <- [1..5]]
c1 + "A^1 prime is even" =  [([2],1),([2],2),([2],3),([2],4
c1 + "Not (A^1 prime is even)" =
   Just [([1],1),([1],2),([1],3),([1],4),([1],5),
         ([3],1),([3],2),([3],3),([3],4),([3],5),
         ([4],1),([4],2),([4],3),([4],4),([4],5),
         ([5],1),([5],2),([5],3),([5],4),([5],5)]
```

- Anaphora predicted to be possible. Diagnosis: too many points
  a. There isn't an[1] even prime, and it$_1$'s not 2.
  b. There's something[1] that's not an even prime, and it$_1$'s not 2.
- Proposal from Anna Alsop: negation works on a per-world basis

$$c + \text{not}(A) = \{\langle g, w\rangle \in c | \neg\exists h : \langle h, w\rangle \in c + A\}$$

- See Elliott 2020 and Kuhn to appear for whether negation ***is*** dynamically closed

Properties of Heim's fragment

- Eliminative: $c + A \subseteq c$
  - The output contains no point that wasn't present in the input
- But not distributive: $c + A \neq \bigcup_{c_n \in c} \{c_n\} + A$
  - Why: the most deeply embedded indefinite must have access to the full local context, to test for novelty

Making Heim's fragment distributive

1. Test the starting context for suitable indices
2. Track a list of unused indices (add State to the monad stack)
3. Existentials presuppose their variable is unused

Refactored implementation available on the repo

- All expressions (once again) fns from g/w pairs to extensions
- In particular, conjunction:

$$[\![\text{A and B}]\!]^{g,w} = \begin{cases} 1 & \text{if } [\![A]\!]^{g,w} = 1 \text{ and } [\![B]\!]^{g,w} = 1 \\ 0 & \text{if } [\![A]\!]^{g,w} = 0 \text{ or } ([\![A]\!]^{g,w} = 1 \text{ and } [\![B]\!]^{g,w} = 0 \\ \# & \text{otherwise} \end{cases}$$

```
type M a = MaybeT ((ReaderT [(G, W)]) (State [Int])) a
and :: M Bool -> M Bool -> M Bool
and ml mr = ml >>= (\l -> if l then mr else return False)
```

- $c + A = \{(g, w) \in c | [\![A]\!]^{g,w} = 1\}$ (presup: $c + A \neq \#$)
- NB: not symmetric, so nor weak nor strong Kleene connective

What the distributive version of Heim's fragment shows

- Distributive version has the same behavior as the original
- Same virtue: stating the dynamics determines truth conditions
- So presup projection is about regulating order of evaluation
- No need to reconceive sentences as CCPs—orthogonal move
- (But maybe epistemic modals are essentially non-distributive)

Dynamic Predicate Logic

1. "The general starting point of the kind of semantics DPL is an instance of, is that the meaning of a sentence does not lie in its truth conditions, but rather in the way it changes (the representation of) the information of the interpreter."

2. "[T]here is a strong correspondence between the dynamic view on meaning, and a basic idea underlying the denotational approach to the semantics of programming languages, viz., that the meaning of a program can be captured in terms of a relationship between machine states... A machine state may be identified with an assignment of objects to variables. The interpretation of a program can then be regarded as a set of ordered pairs of assignments, as the set of all its possible 'input-output' pairs. A pair $\langle g, h \rangle$ is in the iterpretation of a program π, if when π is executed in state g, a possible resulting state is h."

The comparison with programming languages

- Useful model of an idealized semantics/pragmatics interface
- Programs have a timeless meaning
- When uttered—that is, when *run*—programs affect context
- In "imperative" languages, clear separation of content (functions return explicitly typed values) from side effects (input, printing, moving a robotic arm)
- In "functional" languages [declarative?], expressions return a complex value that records both content and scoreboard.
- In a von Neumann architecture, context and pragmatic kinematics perfectly well behaved—nothing will affect the contents of memory cells except the program.

DPL

- Syntax is identical to first order logic. Semantics:

1. $[\![Rt_1 \dots t_n]\!] = \{\langle g, h\rangle \mid h = g \;\&\; \langle [\![t_1]\!]_h \dots [\![t_n]\!]_h\rangle \in F(R)\}$
2. $[\![t_1 = t_2]\!] = \{\langle g, h\rangle \mid h = g \;\&\; [\![t_1]\!]_h = [\![t_2]\!]_h\}$
3. $[\![\neg\phi]\!] = \{\langle g, h\rangle \mid h = g \;\&\; \neg\exists k\colon \langle h, k\rangle \in [\![\phi]\!]\}$
4. $[\![\phi \wedge \psi]\!] = \{\langle g, h\rangle \mid \exists k\colon \langle g, k\rangle \in [\![\phi]\!] \;\&\; \langle k, h\rangle \in [\![\psi]\!]\}$
5. $[\![\phi \vee \psi]\!] = \{\langle g, h\rangle \mid h = g \;\&\; \exists k\colon \langle h, k\rangle \in [\![\phi]\!] \vee \langle h, k\rangle \in [\![\psi]\!]\}$
6. $[\![\phi \to \psi]\!] = \{\langle g, h\rangle \mid h = g \;\&\; \forall k\colon \langle h, k\rangle \in [\![\phi]\!] \Rightarrow \exists j\colon \langle k, j\rangle \in [\![\psi]\!]\}$
7. $[\![\exists x\phi]\!] = \{\langle g, h\rangle \mid \exists k\colon k[x]g \;\&\; \langle k, h\rangle \in [\![\phi]\!]\}$
8. $[\![\forall x\phi]\!] = \{\langle g, h\rangle \mid h = g \;\&\; \forall k\colon k[x]h \Rightarrow \exists j\colon \langle k, j\rangle \in [\![\phi]\!]\}$

Figure 1: DPL semantics

Useful DPL vocabulary

- A formula ϕ is true with respect to an assignment g if g can be extended to h such that $\langle g, h\rangle \in [\![\phi]\!]$.
- A *test* is an eliminative update (hallmark: $\{\langle g, h\rangle | h = g\&...)$
- An expression is *dynamically closed* if the indefinites it contains are not able to serve as antecedents for pronouns outside the expression
  - Negation is a test, so it is dynamically closed
  - Only conjunction and existential quantification are not closed

Implementation—only one monad (List)

```
import Data.List (nub)

replaceAt :: Int -> a -> [a] -> [a]
replaceAt i x xs = take (i-1) xs ++ [x] ++ drop i xs

data Term = Num Int | Var Int deriving (Eq, Show)
data For = Even Term | Odd Term | Prime Term | Precedes Term Term | Follows Term Term | Equals Term Term |
           Not For | And For For | Or For For | If For For | Exists Int For | Forall Int For
           deriving (Eq, Show)

evalTerm :: Term -> [Int] -> Int
evalTerm (Num n) _ = n
evalTerm (Var i) g = if i <= length g then g!!(i-1) else 0

evalFor :: For -> [[Int]] -> [[Int]]
-- the values that begin with "filter" are what G&S call "tests"
evalFor (Even t) gs = filter (\g -> even (evalTerm t g)) gs
evalFor (Odd t) gs = filter (\g -> odd (evalTerm t g)) gs
evalFor (Prime t) gs = filter (\g -> elem (evalTerm t g) [2,3,5]) gs
evalFor (Precedes t1 t2) gs = filter (\g -> (evalTerm t1 g) < (evalTerm t2 g)) gs
evalFor (Follows t1 t2) gs = filter (\g -> (evalTerm t1 g) > (evalTerm t2 g)) gs
evalFor (Equals t1 t2) gs = filter (\g -> (evalTerm t1 g) == (evalTerm t2 g)) gs
evalFor (Not s) gs = filter (\g -> [] == evalFor s [g]) gs
evalFor (And s1 s2) gs = evalFor s2 (evalFor s1 gs)
evalFor (Or s1 s2) gs = filter (\g -> elem g ((evalFor s1 gs) ++ (evalFor s2 gs))) gs
evalFor (If s1 s2) gs = filter (\g -> all (\k -> [] /= (evalFor s2 [k])) (evalFor s1 [g])) gs
evalFor (Exists i s) gs = evalFor s (nub (concat (map (\g -> map (\n -> replaceAt i n g) [1..5]) gs)))
evalFor (Forall i s) gs = filter (\g -> all (\k -> [] /= (evalFor s [k]))
                                            (map (\n -> replaceAt i n g) [1..5])) gs

c1 = [[1],[2],[3],[4],[5]]

s1 = evalFor (Even (Var 1)) c1
s2 = evalFor (Exists 1 (Odd (Var 1))) c1
s3 = evalFor (Exists 1 (Not (Odd (Var 1)))) c1
s4 = evalFor (And (Even (Num 2)) (Odd (Num 3))) c1
s5 = evalFor (Exists 1 (And (Even (Var 1)) (Prime (Var 1)))) c1
s6 = evalFor (Exists 1 (Or (Even (Var 1)) (Prime (Var 1)))) c1
s7 = evalFor (If (And (Even (Var 1)) (Prime (Var 1))) (Equals (Num 2) (Var 1))) c1
s8 = evalFor (Forall 1 (If (Odd (Var 1)) (Prime (Var 1)))) c1
s9 = evalFor (Forall 1 (If (And (Precedes (Var 1) (Num 3)) (Even (Var 1)))
                           (Prime (Var 1)))) c1

-- Donkey anaphora
s10 = evalFor (If (Exists 1 (And (Even (Var 1)) (Prime (Var 1)))) (Equals (Num 2) (Var 1))) c1
s11 = evalFor (Exists 1 (And (Even (Var 1)) (Exists 2 (And (Odd (Var 2)) (Precedes (Var 1) (Var 2))))))
              [[0,0]]
```

Examples

```
c = [[1],[2],[3],[4],[5]]

c + "Even (Var 1)" = [[2],[4]]

c + "Prime (Var 1)" = [[2],[3],[5]]

c + "Exists 1 (Even (Var 1))" = [[2],[4]]

Not eliminative:
[[3]] + "Exists 1 (Even (Var 1))" = [[2],[4]]

Indefinites can antecede pronouns outside their scope:
c + "(And (Exists 1 (Even (Var 1))) (Prime (Var 1)))" =
[[2]]

Information stored in a variable position can be clobbered:
c + "(And (Prime (Var 1)) (Exists 1 (Even (Var 1))))" =
[[2],[4]]
```

Donkey anaphora

If an^1 even number precedes an^2 odd number, it$_2$ follows it$_1$.

```
                                    Local contexts:
[[0]] +
"(If
   (Exists 1                        1,2,3,4,5
     (And (Even (Var 1))            2,4
          (Exists 2                 21,22,23,24,25,41,42,43,44,45
            (And (Odd (Var 2))      21,23,25,41,43,45
                 (Precedes (Var 1) (Var 2)))))) 23,25,45
   (Follows (Var 2) (Var 1)))" =
[[0]]
```

Key theorem of DPL

The antecedent scope of an existential extends arbitrarily far:

$$\exists x \phi \wedge \psi \simeq \exists x [\phi \wedge \psi]$$

1. A^x man entered and he$_x$ sat down.
2. If a^x farmer owns a^y donkey then she$_x$ beats it$_y$.

- BTW, holds for Heim too.

Properties of DPL (reversed from Heim):

- Distributive: $c + A = \bigcup_{c_n \in c} \{c_n\} + A$
  - Semantics stated wrt individual assignment inputs
- But not eliminative: $c + A \not\subseteq c$
  - Why: indefinites introduce a fanout of new assignments

Making DPL eliminative (same strategy as for Heim!)

1. Test the starting context for suitable indices
2. Track a list of unused indices
3. Existentials presuppose their variable is unused

- Now every clause can be a test (filter, eliminative)

What the refactoring of DPL shows

- Essentially a fragment of Heim's system
- That's by design—DPL deliberately sets out to cover same data
- Some choice points for OG dynamic systems, all independent:
  1. Track worlds? (Heim)
  2. Track presuppositions? (Heim)
  3. Indefinites presuppose variability, versus fanout? (Heim)
  4. Negation/Every/If dynamically closed? (DPL)

(3) allows both eliminativity and distributivity.

Challenge question: How to make negation dynamically closed in an eliminative system? Heim's negation eliminates *too few* points. Identifying the relevant points requires figuring out which variables were used by indefinites in the prejacent. Hint1: reason with least-unused-index. Hint2: read Dekker's 1994 PLA, Beaver 2001.

Order of evaluation

- Matt last week (my paraphrase):
  - "I don't see how order of evaluation can be separated from processing."
- Heim's conjunction: c + "A and B" = (c + A) + B
  - Combine A and B in some clever way that then combines with c

1. Go abstract: the untyped lambda calculus
2. Plotkin's 1975 CPS call by name transform—magic!
3. Return to natural language

Order of evaluation in the untyped lambda calculus

- Notes from chapter 12 of Barker and Shan 2014 on repo
- The lambda calculus has three types of terms:
  1. x Variable
  2. $\lambda x M$ Abstract
  3. MN Application
- Key axiom of the lambda calculus: beta reduction
  - $(\lambda x M)N \rightsquigarrow_\beta M[x \mapsto N]$
- The lambda calculus is a good model of computation
  - Turing complete
  - Evaluation consists of successive applications of beta reduction
- Computational behavior depends on order of evaluation
  - $I = \lambda x x, \omega = (\lambda x.xx), \Omega = \omega\omega$
  - $(\lambda x.xx)(\lambda y.yy) \rightsquigarrow_\beta (\lambda y.yy)(\lambda y.yy) \rightsquigarrow_\beta \ldots$
  - $(\lambda x I)\Omega \rightsquigarrow_\beta I$
  - $(\lambda x I)\Omega \rightsquigarrow_\beta (\lambda x I)\Omega \rightsquigarrow_\beta (\lambda x I)\Omega \ldots$

Plotkin's 1975 CPS call by name transform—magic!

```
[*] :: Term -> Term
[x]    = x
[\xM] = \k.k(\x[M])
[MN]   = \k.[M](\m.m[N]k)  -- hides N in arg position
```

- Plotkin's guarantees:
  - for all terms t, t $==$ [t]I
  - [t] can only be evaluated using a leftmost-redex strategy
  - The order of evaluation of the transformed term is independent of the properties of the evaluator
  - How it works: there is at most a single redex at any stage of computation of the transformed term, so there is at most one choice for how to proceed
  - Different evaluation order regimes require different transforms

Example:

$$
\begin{aligned}
([(\lambda x\mathbf{I})\Omega]\mathbf{I}) &= ((\underline{(\lambda\kappa.(\lambda\kappa.\kappa(\lambda x[\mathbf{I}]))(\lambda m.m[\Omega]\kappa))}\mathbf{I}) \\
&= (\underline{(\lambda\kappa.\kappa(\lambda x[\mathbf{I}]))}(\lambda m.m[\Omega]\mathbf{I})) \\
&= (\underline{(\lambda m.m[\Omega]\mathbf{I})}(\lambda x[\mathbf{I}])) \\
&= (((\lambda x[\mathbf{I}])[\Omega])\mathbf{I}) \\
&= (((\lambda x[\lambda xx])[\Omega])\mathbf{I}) \\
&= (((\lambda x(\lambda\kappa.\kappa(\lambda x[x])))[\Omega])\mathbf{I}) \\
&= (((\underline{\lambda x(\lambda\kappa.\kappa(\lambda xx))})[\Omega])\mathbf{I}) \\
&= (\underline{(\lambda\kappa.\kappa(\lambda xx))}\mathbf{I}) \\
&= (\underline{\mathbf{I}}(\lambda xx)) \\
&= (\lambda xx)
\end{aligned}
$$

What this means for natural language semantics

- It is possible to design competence grammars that do not leave order of evaluation to the processing component
- Call by name guarantees that the leftmost redex will be evaluated first
- Generalizing to richer logics, in $(c + A) + B$, $c + A$ will be evaluated first.
- Whatever side effects A has (introducing discourse referents, restricting the local context set, etc.), they will be in place before B is evaluated
- See, e.g., Barker and Shan 2014 for an example of a competence grammar in which evaluation order is regulated to be left to right

(Unsatisfied. What do I still need to do to make this clear?)

The semantics/pragmatics interface

- Yeah, accommodation. How do we hook up a dynamic semantics to a context that is ever-changing even as the evaluation unfolds?
- Beaver 2001: terrific book spelling out in detail the program sketched in Heim 1983.

Beaver's Information Sets (pp. 238 ff.)

- Ignorance corresponds to more points in the context set
- Presuppositions place a requirement on the context set
- Just as a fact can be unsettled (some points true, some false)
  - . . . which context set is the right one can be unsettled
  - if a speaker presupposes something we didn't expect,
  - we had the wrong context set
- So: maintain a *set* of context sets, one for each way the context could be.
- Discard those that produce presupposition violations

"This idea is easily formalised. If states are to be maintained in parallel, then updating must be framed in terms of sets of states, where each state in the set corresponds to a possibly correct model of the speaker's assumptions about the common ground."

- cf. Champollion: imagine points are complete contexts. . .

- Context set: where are we in logical space
- Set of context sets: where are we in epistemic space

Objection 1: is this practical?

- How can we possibly anticipate which information states we might need?
- Every proposition the speaker might choose to presuppose?
- Well, how do we know which worlds to put in the context set?

Objection 2: insufficiently dynamic

- There are reasons to think that context can change (for non-linguistic reasons) in the middle of an utterance in ways that affect evaluation. This scheme does not provide a mechanism for mid-utterance adjustment of the context.

Internally dynamic vs. externally dynamic

Zhuoye Zhao: "On the one hand the semantics defined for DPL builds in some kind of (local) context change to capture anaphoricity, by providing a more rigid interpretation for variables; on the other hand the effect can be achieved in a way that seems really detached from a global context (can be translated into a static PL formula etc.). So does DPL face similar questions as Heim's semantics?"

- Internally dyanamic: dynamics in the service of arriving at appropriate truth conditions (more generally, appropriate content)
- Externally dynamic: dynamics that attempt to characterize the update effect on context
- What happens when we add worlds (intensionality) to DPL?

Preview of GSV: The Broken Vase

- There are three children: Alice, Bob, and Carl. Context:
  - One of them broke a vase.
  - Alice is known to be innocent.
  - Someone is hiding in the closet.

1. Someone^x is in the closet. They$_x$ might be guilty.
2. Someone^x is in the closet who$_x$ might be guilty.
3. $(\exists x.\mathsf{closet}(x)) \wedge (\Diamond \mathsf{guilty}(x)) \not\equiv \exists x(\mathsf{closet}(x) \wedge \Diamond \mathsf{guilty}(x))$

- After update with (1)/(2), what do we know about whether Alice in the closet?
- Oh yeah, the pegs: not used in GSV's fragment