GOTHAM (2019): DOUBLE NEGATION, EXCLUDED MIDDLE & ACCESSIBILITY IN DYNAMIC SEMANTICS[1]

PATRICK D. ELLIOTT

JUNE 17, 2020

[1] Thanks to Simon Charlow and Keny Chatain for useful discussion.

Download this handout https://keybase.pub/patrl/handouts/semtri-gotham.pdf

1 *Accessibility issues*

Famously, classic Dynamic Semantics (DS) gives rise to a notion of *accessibility* that accounts for the (im)possibility of anaphora in the following cases.

(1) John owns a car^x. It$_x$ is parked in a weird place.

(2) *John doesn't own a car^x. It$_x$ is parked in a weird place.

Classical DS accounts for this by making negation a *destructive* operation – any discourse referents introduced in the scope of negation are obliterated, never to be seen again.

The straightforward prediction is that, in a dynamic setting, unlike in a classical setting, double-negation elimination is *not* a valid rule of inference.

Concretely, this gives rise to problems in cases like Partee's famous bathroom sentences:

(3) a. It's not that there's *no* bathroom^x; it$_x$'s upstairs.
  b. Either there is no bathroom^x, or it$_x$'s upstairs.

Question 1: is there a way to preserve the results of classical DS while (re-)validating double negation elimination?

Question 2: Do we *really* want to validate double negation elimination? Gotham will claim that this isn't *quite* right, on the basis of the oddness of sentences like the following, in a context where people are typically assumed to have more than one shirt.

(4) #John doesn't not have a shirt. It's in his closet.

Roadmap:

- A (slightly idiosynratic) crash course in classical DS (i.e., Heim 1982, Groenendijk & Stokhof 1991)
- Issues surrounding accessibility and previous accounts.
- Gotham's *excluded middle* account.
- The status of negation in effectful dynamics.
- Reasons to believe that negation is externally dynamic.

1.1 A crash course in DS

Sentences denote what I'll call "dynamic programs" — *functions from (input) assignments to sets of (output) assignments.*[2]

[2] Assignments here are functions from a stock of variables $V \subseteq \mathbb{N}$ to individuals.

The type of a dynamic program:

(5) $\mathsf{T} \coloneqq \mathsf{g} \to \{\,\mathsf{g}\,\}$

This type encodes both *input-output asymmetry* and *indeterminacy* – the two signature properties of a dynamic system.

Predicates denote functions from individuals to tests.[3]

[3] A test is a trivial dynamic program – Given an input assignment g, a test returns $\{\,g\,\}$ if certain conditions are satisfied, and $\varnothing$ otherwise.

(6) $[\![\text{left}]\!] \coloneqq \lambda xg\,.\begin{cases}\{\,g\,\} & \text{left } x \\ \varnothing & \text{otherwise}\end{cases}$ $\quad \mathsf{e} \to \mathsf{T}$

Names denote individuals:

(7) $[\![\text{John}]\!] \coloneqq \mathsf{j}$ $\quad \mathsf{e}$

Pronouns denote functions from dynamic predicates to dynamic programs:

(8) $[\![\text{he}_1]\!] \coloneqq \lambda kg\,.\,k\,g_1\,g$ $\quad (\mathsf{e} \to \mathsf{T}) \to \mathsf{T}$

Taken together, this means that sentences about individuals/pronouns denote *tests*.

(9) $$[\![\text{John left}]\!] = [\![\text{John}]\!]\ [\![\text{left}]\!] = \lambda g\,.\begin{cases}\{g\} & \mathsf{j\ left}\\ \varnothing & \text{otherwise}\end{cases} \qquad \mathsf{T}$$

If John did in fact leave, then feeding an assignment into the resulting dynamic program returns the singleton set containing that assignment. If he didn't leave, it returns the empty set for every assignment.

As for pronouns, the resulting dynamic program takes an assignment and returns the singleton set containing that assignment, just in case the assignment maps the variable (here 1) to someone that left; otherwise, it returns ∅.

(10) $$[\![\text{he}_1\ \text{left}]\!] = [\![\text{he}_1]\!]\ [\![\text{left}]\!] = \lambda g\,.\begin{cases}\{g\} & g_1\ \mathsf{left}\\ \varnothing & \text{otherwise}\end{cases} \qquad \mathsf{T}$$

Indefinites, like pronouns, denote functions from dynamic predicates to dynamic programs; whereas pronouns simply feed in an individual relative to an input assignment, indefinites *trigger random assignment*.

(11) $$[\![\text{someone}^1]\!] \coloneqq \lambda k g\,.\bigcup_{x:\mathsf{e}} k\ x\ g^{[1\to x]} \qquad (\mathsf{e}\to\mathsf{T})\to\mathsf{T}$$

(12) $$[\![\text{someone}^1\ \text{left}]\!] = [\![\text{someone}^1]\!]\ [\![\text{left}]\!] = \lambda g\,.\{\,g^{[1\to x]} \mid \mathsf{left}\ x\,\} \qquad \mathsf{T}$$

That is to say, a dynamic program involving an indefinite indexed 1, when fed in an assignment g returns a set of gs reflecting the different ways in which g could be updated in accordance with a given constraint.

Let's say that Abed and Shirley left, but Troy didn't (12) will return assignments where 1 gets mapped to a leaver:

$$\lambda g\,.\{\,g^{[1\to\mathsf{abed}]}, g^{[1\to\mathsf{shirley}]}\,\}$$

In dynamic semantics program conjunction (;) is a matter of getting the plumbing right — the assignments outputted by the first conjunct are piped, one by one, into the second conjunct, and the results are gathered together.

(13) $m \,;\, n \coloneqq \lambda g\,.\, \bigcup_{g' \in m\, g} n\, g'$ $\qquad (;) : \mathsf{T} \to \mathsf{T} \to \mathsf{T}$

This allows the modified assignments outputted by a program involving an indefinite to be fed into an input-sensitive program involving a pronoun, pointwise.

Dynamic binding obtains because this setup renders a subsequent conjunct to be sensitive to the output of a previous conjunct.

(14) $[\![\text{someone}^1 \text{ walked in. He}_1 \text{ sat down}]\!]$
$= [\![\text{someone}^1 \text{ walked in}]\!] \,;\, [\![\text{he}_1 \text{ sat down}]\!]$
$= \lambda g\,.\, \bigcup_{g' \in \{ g^{[1\to x]} \mid \text{walked-in } x \}} \{ g'' \mid g' = g'' \wedge \text{sat-down } g'_1 \}$
$= \lambda g\,.\, \{ g^{[1\to x]} \mid \text{walked-in } x \wedge \text{sat-down } x \}$

As far as the expressive power of the system goes, it's useful to note that, if we put indefinites to one side, we can recast dynamic programs as (partial) functions from assignments to assignments, and lose no distinctions – treating the output as a *set* of assignments is only necessary to model the semantic contribution of indefinites.[4]

[4] In other words, in the absence of indefinites, we could think of dynamic programs as functions from assignments to assignments:

(15) $\mathsf{T} \coloneqq \mathsf{g} \to \mathsf{g}$

...and predicates are functions from individuals to tests, where tests are just partial identity functions...

(16) $\text{left} \coloneqq \lambda x g : \text{left } x\,.\, g \qquad \mathsf{e} \to \mathsf{T}$

...and program conjunction as vanilla function composition...

(17) $m \,;\, n \coloneqq n \circ m$

Pronouns can receive the same definition. Something like dynamic binding is still possible; we can define a function n to convert an individual into a binder. This leaves no room for indeterminacy however – every dynamic program involving a binder will take an assignment g and either be undefined or return a deterministically modified assignment.

(18) $x^1 \coloneqq \lambda k g\,.\, k\, x\, g^{[1\to x]}$

1.2 *Negation in* DS

Negation in DS is defined as follows: it takes a dynamic program, and returns a program that takes an assignment g, and returns $\{ g \}$ if feeding g into the input program returns $\varnothing$, and $\varnothing$ otherwise.

(19) $\text{not } m \coloneqq \lambda g\,.\, \begin{cases} \{g\} & m\, g = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \qquad \text{not} : \mathsf{T} \to \mathsf{T}$

For any input program, dynamic negation therefore *always* returns a test. This guarantees that any Discourse Referents (DRS) induced by indefinites in the scope of negation are *obliterated*.

To illustrate, recall that the program denoted by "someone left" maps assignments to updated assignments like so:

(20) $[\![\text{someone}^1 \text{ left}]\!] = \lambda g \,.\, \{\, g^{[1\to\text{abed}]}, g^{[1\to\text{shirley}]} \,\}$ $\qquad \mathsf{T}$

What happens when we add dynamic negation into the mix? The resulting program is going to take assignments, and check that feeding them into the program above results in the empty set.

In a world where there are two leavers, this will correctly return the empty set. In a world where there are no leavers, this will be a test.

(21) $$[\![\text{it's not that case that someone}^1 \text{ left}]\!] = \mathsf{not}\,([\![\text{someone}^1]\!] \;\; [\![\text{left}]\!]) = \lambda g \,.\, \begin{cases} \{g\} & \{\, g^{[1\to x]} \mid \mathsf{left}\; x \,\} = \varnothing \\ \varnothing & \text{otherwise} \end{cases}$$

This means that an existential in the scope of negation is *inaccessible* as an antecedent to a subsequent pronoun.

Operators, like negation, which take dynamic programs as arguments and always return tests are called *externally static*.

In DS, disjunction is also taken to be externally static (at least, in Dynamic Predicate Logic (DPL)):

(22) $$m \,\underline{\vee}\, n \coloneqq \lambda g \,.\, \begin{cases} \{g\} & m\, g \cup n\, g \neq \varnothing \\ \varnothing & \text{otherwise} \end{cases} \qquad (\underline{\vee}) : \mathsf{T} \to \mathsf{T} \to \mathsf{T}$$

Also, universal quantification:

(23) $$[\![\text{everyone}^1]\!] \coloneqq \lambda k g \,.\, \begin{cases} \{g\} & \forall x [k\; x\; g^{1\to x} \neq \varnothing] \\ \varnothing & \end{cases} \qquad (\mathsf{e} \to \mathsf{T}) \to \mathsf{T}$$

Externally static operators are characterized as follows - they take *any* dynamic program, including tests, and return tests. It follows, therefore that double negation elimination will not be a valid inference rule.

Let's just briefly see why – a sentence with an indefinite under two negations is a program that takes an input assignment g, and returns $\{\, g \,\}$ only if feeding g into the embedded program returns the empty set. This will hold just in case there is at least one leaver.

This gets the right results in terms of truth conditions, and furthermore predicts that, as soon as an indefinite is under one negation, it is never again accessible for subsequent pronouns.

(24) ⟦it's not the case that nobody left⟧

$$= \text{not}\,(\text{not}\,([\![\text{someone}^1]\!] \quad [\![\text{left}]\!]))$$

$$\lambda g\,.\begin{cases}\{g\} & (\text{not}\,([\![\text{someone}^1]\!] \quad [\![\text{left}]\!]))\ g = \varnothing\\ \varnothing & \end{cases}$$

1.3 The problem of double negation

Gotham's paper is concerned with the fact that classical DS apparently makes bad predictions in this domain, and how to minimally change the underlying logic in order to fix this.

First off, as is well known, it's simply false that existentials in the scope of two negations are *never* accessible for subsequent pronouns.

(25) I'ts not true that John doesn't own a car^1.
It$_x$'s just parked in a weird place.

In DS, the first conjunct will denote a test. Dynamic conjunction feeds the output assignment into the second conjunct, and the pronoun is interpreted relative to the global input – in other words, it's free.

A similar problem arises with disjunction:

(26) Either John doesn't own a car^1, or it$_1$'s parked in a weird place.

Assuming that the existential takes narrow scope wrt disjunction, this will have the following meaning in DS:

(27) ⟦Either John doesn't own a car^1 or it$_1$'s parked in a weird place⟧

$$= (\text{not}\,([\![\text{John owns a car}^1]\!]))\ \underline{\vee}\ ([\![\text{it}_1]\!] \quad [\![\text{parked in a weird place}]\!])$$

$$= \lambda g\,.\begin{cases}\{g\} & [\![\text{John doesn't own a car}]\!]\ g \cup [\![\text{it parked weird place}]\!]\ g \neq \varnothing\\ \varnothing & \end{cases}$$

Disjunction, as well as being externally static is *internally static*. A binary operator over dynamic programs as internally static if a existential in one of the juncts isn't accessible to a pronoun in the other junct.

Gotham brings out this property of disjunction in DS in a nice way. The following is equivalent to (26) when rendered into First-order Logic (FoL):[5]

(28) Either John doesn't own a car^1,
[or John does own a car^1, and it$_1$'s parked in a weird place].

[5] This is because $\neg p \vee q$ is equivalent to $p \rightarrow q$. $\neg p \vee (p \wedge q)$ is therefore equivalent to $p \rightarrow (p \wedge q)$. It's easy to see that $p \rightarrow q$ is equivalent to $p \rightarrow (p \wedge q)$.

Because double negation elimination is valid in FoL, this also turns out equivalent to the following (again, only when rendered into FoL):

(29) Either John doesn't own a car^1,
or [John doesn't not own a car^1, and it$_1$'s parked in a weird place].

In a DS setting however, (26) is equivalent to (29) but *not* (28). This is because, in (28), but not (29), the pronoun is bound.

1.4 Uniqueness effects

We've seen a couple of cases where it seems like plain old FoL makes superior predictions to DS – do we want to somehow validate double negation elimination?

It turns out there's a sense in which neither FoL nor DS get things quite right.[6]

[6] The judgments here are Gotham's.

(30) a. ?? It's not true that John doesn't own a shirt. It's in the wardrobe.
b. ?? Either John doesn't own a shirt, or it's in the wardrobe.

Gotham claims that (30) sound odd because they give rise to the implication that *If John owns a shirt, then he owns exactly one.*

Crucially, this is not taken to be an inference associated with the positive counterpart:[7]

(33) John owns a shirt. It's in the wardrobe.

[7] I actually think that (33) at least implies that the speaker doesn't know that John has more than one shirt. Surprisingly, the following sounds OK to me in a context where the speaker does know this:

(31) John owns a shirt. They're in the wardrobe.

Moreover:

(32) It's not true that John doesn't own a shirt. They're in the wardrobe.

It seems like the source of the uniqueness inference must somehow be connected to the singular pronoun.

Gotham puts the question of the nature of the putative uniqueness inference to one side – presupposition, scalar implicature, conventional implicature, etc.

It's kind of hard to test actually, since the usual tests involve embedding under Downward Entailing (DE) operators, which are externally static, and so may interfere in other ways. Any suggestions are welcome.

2 *Previous Accounts*

As Gotham points out, there have already been attempts to address this in the literature.

2.1 *Separating negation from dynamic closure*

In DS, truth relative to an assignment can be retrieved by existentially closing the output set.

(34) $m^{\downarrow} \coloneqq \lambda g\,.\,\exists g'[m\,g]$ $\quad \texttt{T} \to \texttt{g} \to \texttt{t}$

If we close a dynamic program involving an existential, we get standard static truth conditions, in fact.

(35) takes an assignment, and returns true just in case there's at least one way of mapping 1 to a leaver, which will only be possible if there *is* such a leaver.

(35) $[\![\text{someone}^1 \text{ left}]\!]^{\downarrow} = \left(\lambda g\,.\,\{\,g^{[1\to x]} \mid \mathsf{left}\;x\,\}\right)^{\downarrow} = \lambda g\,.\,\exists g' \in \{\,g^{[1\to x]} \mid \mathsf{left}\;x\,\}$

We can lift this back into a dynamic program by using the following function, which lifts any static proposition into the corresponding test (($\uparrow$) is therefore externally static):[8]

(37) $p^{\uparrow} \coloneqq \lambda g\,.\begin{cases}\{\,g\,\} & p\,g\\ \varnothing\end{cases}$ $\quad (\texttt{g} \to \texttt{t})$

[8] Note that the DPL closure operator ! discussed by Gotham is basically equivalent to the composition of these two operations:

(36) $! = {\downarrow\uparrow}$

If we lower and then lift a program with an existential, we get a dynamic program that is a *test*, with existential conditions.

(38) $[\![\text{someone}^1 \text{ left}]\!]^{\downarrow\uparrow} \coloneqq \lambda g\,.\begin{cases}\{\,g\,\} & \exists g' \in \{\,g^{[1\to x]} \mid \mathsf{left}\;x\,\}\\ \varnothing & \text{otherwise}\end{cases}$

One way of accounting for the double negation facts is to state a basic entry for negation as follows:

(39) $\neg\, m \coloneqq \lambda g\,.\,\{\,g' \mid g' \notin m\,g\,\}$

Let's see what happens if we apply (39) to a program with an existential. Imagine again that Shirley and Abed are leavers, "someone left" will denote the following:

(40) $\lambda g \,.\, \{ g^{1 \to \text{abed}}, g^{[1 \to \text{abed}]} \}$

Now let's apply ¬ once:

(41) $\lambda g \,.\, \{ g' \mid g' \notin \{ g^{1 \to \text{abed}}, g^{[1 \to \text{abed}]} \} \}$

This will take an assignment g, and simply give back assignments that don't map 1 to a leaver. Applying ¬ again gives back our original value:

(42) $\lambda g \,.\, \{ g' \mid g' \notin \{ g'' \mid g'' \notin \{ g^{[1 \to \text{abed}]}, g^{[1 \to \text{shirley}]} \} \} \}$
$\equiv \lambda g \,.\, \{ g^{[1 \to \text{abed}], g^{[1 \to \text{shirley}]}} \}$

Just so long as negation is interpreted as ¬, then, double negation elimination is validated. We can even retrieve the original entry for dynamic negation by closing our dynamic program with ⇃↾, negating the result, and then ensuring the result is a test with the operator •:

(43) $\bullet\, m \coloneqq \lambda g \,.\, \begin{cases} \{g\} & g \in m\, g \\ \varnothing & \text{otherwise} \end{cases}$

(44) $\mathsf{not} \coloneqq \bullet \circ (\neg) \circ \downharpoonleft\!\upharpoonright$

As pointed out by Gotham however, if negation must be interpreted as ¬ in order to account for the double negation cases, why can't it always be interpreted as ¬? If this were possible, our original example, repeated below, would have a reading: *It's not true that John owns a car that is parked in a weird place.*

(45) John doesn't own a car^1. It$_1$ is parked in a weird place.

That's because, outputs of the first conjunct must map 1 to non-cars; the conjoined program can therefore only return a non-empty-set if only non-cars are parked in weird places, just so long as the indefinite and the pronoun are coindexed.

Furthermore, this wouldn't account for the observed uniqueness effect, since it essentially validates double negation elimination.

Gotham also cursorily discusses a proposal by Krahmer and Muskens (1995) to keep track of both positive and negative extensions. We'll consider a variation on this later.

3 Gotham's proposal

Gotham's proposal is inspired by *intuitionistic logic*, in which adding the principle of the excluded middle as an axiom gives one back classical logic:

(46) The principle of the excluded middle
$\phi \vee \neg \phi$

The question he poses is: can adding (someone like) the principle of the excluded middle to classical DS give allow us to get back to a system which validates double negation elimination?

Gotham argues that the answer is *yes*, however, the principle of the excluded middle should not be stated in terms of dynamic disjunction, but rather in terms of what Groenendijk & Stokhof (1991) call *program disjunction*, defined below.

(47) Program disjunction (def.)
$m \uplus n \coloneqq \lambda g\,.\, m\, g \cup n\, g$

Unlike DS disjunction, G&S's program disjunction is externally dynamic.

In DS, $m \uplus \textsf{not}\ m$ will be trival, but trivial programs can be semantically distinct, depending on their anaphoric potential – $m \uplus \textsf{not}\ m$ is a trivial program that inherits the anaphoric potential of m.

Gotham's innovation is the claim that in DS, the discourse is interpreted in the context of a specific tautology, namely instances of the excluded middle with program disjunction.

(48) (Either John has no shirt^1 or he has a shirt^1 and) It's not the case that John doesn't have a shirt^1.

(49) (Either John owns a car^1 or John doesn't own a car^1 and) either John doesn't own a car^1 or it$_1$'s parked in a strange place.

If John owns shirts a and b, then "Either John has no shirt, or he has a shirt" will denote the following:

(50) $[\![\text{John has no shirt}^1]\!] \uplus [\![\text{John has a shirt}^1]\!]$
$= \lambda g\, .\, (\varnothing \cup \{\, g^{[1\to a]}, g^{[1\to b]} \,\})$
$= \lambda g\, .\, \{\, g^{[1\to a]}, g^{[1\to b]} \,\}$

If John doesn't own any shirts, it will return a test:

(51) $[\![\text{John has no shirt}^1]\!] \uplus [\![\text{John has a shirt}^1]\!]$
$= \lambda g\, .\, (\{\, g \,\} \cup \varnothing)$
$= \lambda g\, .\, \{\, g \,\}$

The output of (not m) $\uplus$ m is *always* non-empty, but depending on how the world is, it has the ability to introduce discourse referents, which will be accessible for the purpose of subsequent conjuncts.

4 *Uniqueness*

The above of course doesn't account for the uniqueness effect.

Gotham's claim: "while a simple indefinite *a P* licenses subsequent pronouns, one under double negation only does so on the assumption that there is exactly one *P*"[9]

[9] How do we square this with the following datapoint?

(52) It's not true that John doesn't own a shirt, they're in his closet.

Let's recall what program disjunction does for an existential statement and its negation:

(53) $[\![\text{someone}^1\text{ left}]\!] \uplus [\![\text{nobody}^1\text{ left}]\!] = \lambda g\, .\, \begin{cases} \{\, g^{[1\to x]} \mid \text{left } x \,\} & \exists x[\text{left } x] \\ \{\, g \,\} \end{cases}$

In order to capture uniqueness, we instead want program disjunction to deliver the following:

(54) $\lambda g\, .\, \begin{cases} \{\, g^{[1\to x]} \mid \text{left } x \,\} & \exists ! x[\text{left } x] \\ \{\, g \,\} \end{cases}$

Note that, due to the uniqueness requirement, this will result in a program that is *always* deterministic with respect to 1!

The kind of privileged tautology we now get, we can refer to as Unique Excluded Middle (UEM).

(55) The Unique Excluded Middle (UEM) (def.)

$$\upharpoonleft m \coloneqq \lambda g \, . \begin{cases} m\ g & |m\ g| = 1 \\ \{g\} \end{cases}$$

A m that is doubly negated is interpreted in the context of $\upharpoonleft m$. This gives us the uniquess effect.

(56) $\upharpoonleft([\![\text{someone}^1 \text{ left}]\!])\,;\,\textsf{not}\,(\textsf{not}\ [\![\text{someone}^1 \text{ left}]\!])$

$$= \left(\lambda g \, . \begin{cases} \{g^{[1\to x]} \mid \textsf{left}\ x\} & |m\ g| = 1 \\ \{g\} \end{cases}\right); \left(\lambda g \, . \begin{cases} \{g\} & g \notin [\![\text{not someone}^1 \text{ left}]\!]\ g \\ \varnothing \end{cases}\right)$$

If exactly one person left, say Shirley, this will map assignments to a single updated assignment:

(57) $\lambda g \, . \, \{g^{[1\to \textsf{shirley}]}\}$

If more than one person left, then the resulting program won't return the empty set, but it *will* be a test, it will simply return the singleton set containing the input, and therefore be equivalent to just the second conjunct.

If nobody left, the program will return the emptyset.

4.1 *Do we really want uniqueness?*

Citing Matt Mandelkern (p.c.), Gotham notes some data that seems to go against the claim that double negation gives rise to a uniqueness effect (this is reminiscent of our previous example: "John doesn't not have a shirt; they're in his wardrobe.").

(58) Either Sue didn't have a drink last night, or she had a second drink right after it.

Gotham doesn't say anything definitive about this, other than expressing mild skepticism about the judgment.

He furthermore notes that we can roll back to the plain old excluded middle (which would predict Matt's judgment here), if this turns out to be true.

5 *Negation cooked a different way: effectful dynamics*

Charlow (2014) developed an elaboration of DS, using techniques from functional programming for modelling side-effects (hence: "effectful dynamic semantics").

This works just like classical DS, only "pure" semantic composition is preserved, and the dynamic side-effects of an expression are modeled explicitly using the `State.Set` type constructor.

(59) `State.Set` (def)
$\mathsf{SS\ a} \coloneqq \mathsf{g} \rightarrow \{ (\mathsf{a}, \mathsf{g}) \}$

Return specifies how to lift an ordinary semantic value into a dynamic setting:
(60) Return (def.)
$g^{\eta} \coloneqq \lambda g \,.\, \{ (x, g) \}$ $\qquad \mathsf{a} \rightarrow \mathsf{SS\ a}$

In an effectful setting, application does pointwise application of the ordinary semantic values, and passes assignments from left-to-right.

(61) Application in `State.Set`
$m \circledast n \coloneqq \lambda g \,.\, \{ (x \mathbin{\mathsf{A}} y, g'') \mid (x, g') \in m\, g \wedge (y, g'') \in n\, g' \}$

$$\begin{cases} \mathsf{SS\ (a \rightarrow b) \rightarrow SS\ a \rightarrow SS\ b} \\ \mathsf{SS\ a \rightarrow SS\ (a \rightarrow b) \rightarrow SS\ b} \end{cases}$$

Indefinites in effectful dynamics:

(62) $[\![\text{someone}^{n}]\!] \coloneqq \lambda g \,.\, \{ (x, g^{[n \rightarrow x]}) \}$ $\qquad \mathsf{SS\ e}$

Application is helped along by lifting anything not inherently effectful.

$$\dfrac{\begin{array}{c}\text{someone}^{n} \\ \vdots \\ \lambda g \,.\, \{ (x, g^{[n \rightarrow x]}) \mid x : \mathsf{e} \} : \mathsf{SS\ e}\end{array} \qquad \dfrac{\begin{array}{c}\text{left} \\ \vdots \\ \lambda x \,.\, \mathsf{left}\ x : \mathsf{e} \rightarrow \mathsf{t}\end{array}}{\lambda g \,.\, \{ (\lambda x \,.\, \mathsf{left}\ x), g \} : \mathsf{SS\ (e \rightarrow t)}}\,\eta}{\lambda g \,.\, \{ (\mathsf{left}\ x, g^{[n \rightarrow x]}) \mid x : \mathsf{e} \} : \mathsf{SS\ t}}\,\circledast$$

In DS, a sentential meaning is a function from an assignment to a set of assignments; in an effectful setting, a sentential meaning is a function from an assignment, to a set of truth-value-assignment pairs.

This means that sentential meanings in effectful dynamics encode *more information* - they keep track of false-tagged assignments as well as true-tagged assignments.

Consider, e.g., a world in which Shirley and Abed left, and Troy didn't leave. The meaning of "someone¹ left" in an effectful setting will be:

(63) $\lambda g\,.\,\{(\top, g^{[1\to\text{shirley}]}), (\top, g^{[1\to\text{abed}]}), (\bot, g^{[1\to\text{troy}]})\}$ $\qquad$ `SS t`

In ᴅs, however, we only keep track of the assignments that are true-tagged:

(64) $\lambda g\,.\,\{g^{[1\to\text{shirley}]}, g^{[1\to\text{abed}]}\}$ $\qquad$ `T`

We can actually write a simple function to get us from an effectful sentential meaning to the corresponding ᴅs sentential meaning:

(65) Function from effectful to ᴅs sentential meanings:
$\lambda m\,.\,\lambda g\,.\,\{g' \mid (\top, g') \in m\ g\}$ $\qquad$ `SS t → T`

This extra information gives us more options in terms of how to define negation.

If we take negation to basically just be truth-conditional negation, we can define a function to *map* negation through the `State.Set` scaffolding, into the contained value.[10]

[10] *map* is defined as follows:

(66) $f^{map} m \coloneqq$
$\lambda g\,.\,\{(f\ x, g') \mid (x, g') \in m\ g\}$
`map :=`

(67) $\text{not}^{map}\ m \coloneqq \lambda g\,.\,\{(\neg\ t, g') \mid (t, g') \in m\ g\}$

If we apply not^{map} to "someone¹ left", all it does is flip the true-tagged and false tagged assignments, so if Shirley and Abed left, and Troy didn't leave, we get:

(68) $\text{not}^{map}\ (\lambda g\,.\,\{(\top, g^{[1\to\text{shirley}]}), (\top, g^{[1\to\text{abed}]}), (\bot, g^{[1\to\text{troy}]})\})$
$= \lambda g\,.\,\{(\bot, g^{[1\to\text{shirley}]}), (\bot, g^{[1\to\text{abed}]}), (\top, g^{[1\to\text{troy}]})\}$

In other words, the resulting meaning introduces a discourse referent that *didn't leave*; mapping propositional negation into a sentence with an indefinite simply allows the indefinite to outscope negation.

Simply mapping propositional negation then, gives us a pretty sensible meaning for the following sentence:

(69) It's not the case that someone¹ left. $\qquad$ $\exists > \neg$

Idea: we can make sense of negation in effectful dynamics if we first evaluate the scope of the indefinite *before* mapping in negation.

The standard way to close off the scope of an existential in DS is by existentially closing the output assignment, and returning a test. We can try the same thing in an effectful setting.

(70) `State.Set` closure (first attempt) $m^{\downarrow\uparrow} \coloneqq \lambda g \,.\, \{(\exists(\top, g') \in m\, g, g)\}$
`SS t → SS t`

This will essentially accomplish the same thing as classical dynamic negation. Any discourse referents are obliterated. We can reconstruct classical DS negation in an effectful setting as the composition of closure and negation.

(71) $\mathsf{not}^{map}\ [\![\text{someone left}]\!]^{\downarrow\uparrow\, 1}$
$= \mathsf{not}^{map}\ (\lambda g \,.\, \{(\exists(\top, g') \in \{(x\ \mathsf{left}, \mathsf{g}^{[1\rightarrow x]})\}, g)\})$

If there is at least one-leaver, then (5) will return a test with a true-tagged output. If there aren't any leavers, then (5) will return a test with a false-tagged output.

This will face exactly the same issues as classical DS negation, wrt double negation. So, let's try something else. We can define a closure operator that preserves only true-tagged assignments – false-tagged assignments are simply discarded. If there are no true-tagged assignments, the result will be a test with a false-tagged output.

(72) `State.Set` closure (second attempt)
$$m^{\downarrow\uparrow} \coloneqq \lambda g \,.\, \begin{cases} \{(\top, g') \mid (\top, g') \in m\ g\} & \exists(\top, g') \in m\ g \\ \{(\bot, \mathsf{g})\} & \text{otherwise} \end{cases} \quad (\downarrow\uparrow) : \texttt{SS t} \rightarrow \texttt{SS t}$$

Revised dynamic closure reduces information, but doesn't eliminate discourse referents (its externally dynamic):

(73) $[\![\text{someone}^1\ \text{left}]\!]^{\downarrow\uparrow}$ `SS t`
$= (\lambda g \,.\, \{(\top, \mathsf{g}^{[1\rightarrow \mathsf{shirley}]}), (\top, \mathsf{g}^{[1\rightarrow \mathsf{abed}]}), (\bot, \mathsf{g}^{[1\rightarrow \mathsf{troy}]})\})^{\downarrow\uparrow}$
$= \lambda g \,.\, \{(\top, \mathsf{g}^{[1\rightarrow \mathsf{shirley}]}), (\top, \mathsf{g}^{[1\rightarrow \mathsf{abed}]})\}$

Miraculously, if we compose our revised dynamic closure operator with mapped truth-conditional negation, the result is externally static! True-tagged modified assignments become false tagged.

(74) $\textsf{not}^{map}\ (⟦\text{someone}^1\ \text{left}⟧^{\downarrow\uparrow}) = \lambda g\ .\ \{(\bot, g^{[1\to\textsf{shirley}]}), (\bot, g^{[1\to\textsf{abed}]})\}$

Now we see the advantages of effectful dynamics – applying negation *again* reverses the polarity of the assignments, making the referents available again.

(75) $\textsf{not}^{map}\ (\textsf{not}^{map}\ (⟦\text{someone}^1\ \text{left}⟧^{\downarrow\uparrow})) = \lambda g\ .\ \{(\top, g^{[1\to\textsf{shirley}]}), (\top, g^{[1\to\textsf{abed}]})\}$

`SS tt`

We've now achieved something like double negation elimination in effectful dynamics! All we needed was a modified closure operator that preserves information about true assignments.

This of course doesn't get the uniqueness effect associated with double negation. The suggestion I'd like to make is that we should nuance the *closure* operator while maintaining a standard truth-conditional entry for negation.

(76) `State.Set` closure (third attempt)

$$m^{\downarrow\uparrow} := \lambda g\ .\ \begin{cases}\{(\top, \iota g'\{g' \mid (\top, g') \in m\ g\})\} & \exists(\top, g') \in m\ g\\ \{(\bot, g)\} & \text{otherwise}\end{cases}$$

In plain English, if there are any true-tagged assignments, this closure operator presupposes there is *exactly one*, and returns it. If there are none, it simply returns a test.

We might want to nuance this further to account for the fact that a plural indefinite under double negation leads to reintroduction of a *maximal* discourse referent — In (77), *they*$_1$ picks out all of the shirts that John has, necessarily.

(77) John doesn't have no shirts^1. They$_1$'re in his closet.

We can cash this out in the current setting by (a) making certain natural assumptions about the semantics of a plural indefinite, (b) substituting in a maximality operator, in place of the iota in our closure operation.

(78) $⟦\text{some boys}^1⟧ := \lambda g\ .\ \{(X, g^{[1\to X]}) \mid X \in \textsf{boy}^*\}$

If Troy and Abed left, but Jeff didn't leave, then "some boys^1 left" will have the following meaning:

(79) $\lambda g \, . \, \{ (\top, g^{[1 \to \text{troy}]}), (\top, g^{[1 \to \text{abed}]}), (\top, g^{[1 \to \text{troy} \oplus \text{abed}]}), (\bot, g^{[1 \to \text{Jeff}]}) \}$

We want our closure operator to take the set of true-tagged assignment, and pick out the (unique) "maximal" one. Here's an attempt at defining maximality for assignments. It takes a set of assignments (let's assume that they all have the same domain), returns that g, such that for every index n in the domain, the value of g_n is maximal relative to the return value of the other assignments at n.[11]

[11] Note that this subsumes uniqueness in the case of singular indefinites.

(80) $\text{max}\, G \coloneqq \iota g[g \in G \wedge \forall g' \in G[\exists n[g_n, g'_n \neq \# \rightarrow g'_n \sqsubseteq g_n]]]$

Now we can redefine our closure operator *again* in terms of maximality:

(81) `State.Set` closure (final attempt)

$$m^{\downarrow\uparrow} \coloneqq \lambda g \, . \begin{cases} \{ (\top, \boxed{\text{max}\, g' \mid (\top, g') \in m\, g}) \} & \exists (\top, g') \in m\, g \\ \{ (\bot, g) \} & \text{otherwise} \end{cases}$$

The looming question here is the following — since, on this account, we've decomposed dynamic negation into (a) truth-conditional negation, and (b) dynamic closure, do we find evidence for dynamic closure, and it's associated maximality effect, outside of the domain of negation? Places to look include maximal informativity effects with anaphora[12] and weak vs. strong readings of donkey sentences (see Kanazawa 1994, Chierchia 1995, and others).

[12] Keny Chatain (p.c.) has interesting data on this!

6 *Related issues*

6.1 *Unbounded dynamic scope of definites*

Assumption: sloppy readings necessitate binding:

Thanks to Simon Charlow (p.c.) for pointing out the relevance of these cases.

(82) Abed^1 thinks that Shirley likes him$_1$,
and TROY^2 thinks that she DOESN'T ~~like him$_2$~~.

Moreover, dynamic binding licenses sloppy readings too:

(83) Everyone who talked to SHIRLEY^1 insulted her$_1$,
and everyone who talked to ANNIE^2 did ~~insult her$_2$~~ too.

A problem: if negation closes renders antecedents inaccessible, the following is predicted to lack a sloppy reading, contrary to fact:

(84) Everyone who doesn't know anyone [who is friends with Shirley^1]
gossiped about her$_1$
and everyone who doesn't know anyone [who is friends with ANNIE^2]
did ~~gossip about her$_2$~~ too.

This is because the islandhood of the relative clause means that we can be reasonably sure(?) that the definite is scoping below negation.[13]

[13] There's a potential worry here concerning exceptional scope.

Relatedly, as I've shown in previous presentations, if negation is really externally static, then we lose Chierchia's account of Weak Crossover (WCO) in terms of accessibility.

Perhaps understanding how negation can be externally dynamic with respect to deterministic DRS can help us understand why double-negation gives rise to a uniqueness effect.

References

Charlow, Simon. 2014. *On the semantics of exceptional scope*. Dissertation.

Chierchia, Gennaro. 1995. *Dynamics of meaning: Anaphora, presupposition, and the theory of grammar*. Chicago: University of Chicago Press. 270 pp.

Chierchia, Gennaro. 2020. Origins of weak crossover: When dynamic semantics meets event semantics. *Natural Language Semantics* (28). 23–76.

Gotham, Matthew. 2019. Double negation, excluded middle and accessibility in dynamic semanttics. In Julian J. Schlöder, Dean McHugh & Floris Roelofsen (eds.), *Proceedings of the 22nd Amsterdam Colloquium*, 142–151.

Groenendijk, Jeroen & Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy* 14(1). 39–100.

Heim, Irene. 1982. *The semantics of definite and indefinite noun phrases*. 2011 edition - typesetting by Anders J. Schoubye and Ephraim Glick. University of Massachusetts - Amherst dissertation.