Introduction to Dynamic Semantics

Patrick D. Elliott

25.06.2018

Lecture 1: The Formal Semanticist's Toolkit

Slides

- pdf: https://patrl.keybase.pub/actl2018/lecture1.pdf
- html (support here still sketchy!):
  https://patrickdelliott.com/actl2018notes.html

Roadmap

- An overview of what semanticists are interested in.
- A lightning-fast overview of some basic formal machinery: sets, functions, and lambdas.
- Constructing our first formal language L_1.
- *First Order Logic* (FOL)
- Next week: *Dynamic Predicate Logic* (DPL)

Formal semantics

- This class is going to provide you with an introduction to *formal semantics*.
- What formal semantics is *not*:
    - A theory of the connotations and cultural associations of particular words and phrases.
    - A theory of how we communicate through language.
    - A theory of concepts, and how we acquire them.
- Rather formal semantics is typically concerned with the *logical* (i.e. *formal*) properties of language. Arguably, a theory of formal semantics should inform all of the above.

Formal semantics ii

- But...why isn't this class called *Introduction to Formal Semantics*?
- The particular perspective we're going to work towards is that of *dynamic semantics*, a family of theories developed in the eighties and nineties by Irene Heim, Hans Kamp, Jeroen Groenendijk, Martin Stokhof, Paul Dekker, and others.

Dynamic Semantics

- Dynamic semantics is an approach to formal semantics which provides a unified perspective on anaphora, quantification, and the flow of discourse.
- In the first part of the class, we're going to work towards an analysis of *donkey anaphora*, illustrated by the famous example sentence in (1), using Groenendijk and Stokhof's theory *Dynamic Predicate Logic*.

  (1) every farmer who owns a donkey^{x} hits it$_{x}$

Dynamic semantics ii

- In most introductory classes, what is taught is a strictly *static* semantics, where the unit of analysis is strictly the *sentence*.
- One of the key insights of dynamic semantics is that, not just sentences, but *discourses* can be assigned truth-conditions.

  (2) A man^{x} walked in the room. He$_{x}$ sat down.

- (3) is true iff there is a man x, s.t. x walked in the room and x.

Dynamic semantics iii

- One thing we're going to be thinking through is the relationship between a static semantics a dynamic semantics.
- In today's class, we're going to work towards a formal, static semantics of English, using First Order Logic as our tool.
- But first, some broad perspectives on the study of meaning in natural language.

In the beginning, there was syntax

- It's all about this guy:

Figure 1: Noam Chomsky (right)

In the beginning, there was syntax ii

- (Generative) syntax, when you really get down to it, is the study of an individual language (or idiolect) L. A "language", such as English, is just a useful abstraction over a set of sufficiently similar Ls.
- The question syntacticians ask themselves is: how can we specify what the possible sentences of L are?

Syntactic methodology

- Typically, syntacticians do this by giving a *grammar* of L.
- Depending on your theory, a grammar can be a set of rewrite rules, a recursive procedure for building structured representations, or a combinary logic. It amounts to more-or-less the same thing at the end of day.
- A grammar of L is taken to be an abstract description of What We Know When We Know L.

An aside

- Some (generative) theories of syntax:
    - Lexical Functional Grammar, Combinary Categorial Grammar, Minimalism, ...
- I'm going to try to stay as neutral as possible about which flavour of syntax tastes the best.
- When I *have* to say something about syntax, I'm going to assume a simplified version of Chomsky's Government and Binding theory, since it's the one that most people seem to have been exposed to.
- I'm happy to argue about, e.g., construction grammar, in the pub...not so much in class.

Syntactic intuitions ii

- How do we know what a possible sentence of L is?
  - If we're a native speaker of L, we can consult our own intuitions about the acceptability of a given sentence.
  - We can consult a native speaker of L.
- The vast majority of the raw data syntacticians deal with is *speaker acceptability judgements*, be they collected informally or experimentally.

Meaning as truth-conditions

- To know the meaning of a sentence is to know its *truth conditions*.

Figure 2: Polish mathematician and Logician Alfred Tarski
(src: the Oberwolfach photo collection)

Meaning as truth-conditions

(4) Russians penetrated U.S. voter systems.

- We don't need to know whether or not (4) is *true* to know what it *means*.
- What we do know (roughly), is how the world would have to be for (4) to be true – (4) is *true* if Russians penetrated U.S. voter systems, and *false* otherwise.
- Is this just stating the obvious?

Semantic intuitions

- Let's say we're studying the semantics of a given language L, and we have the syntax of L pretty well-worked out.
- Our task as semanticists is to specify a recursive procedure for mapping sentences of L to *meanings*.
- How do we know what a given sentence of L means? Just as before, we have to rely on native speaker intuitions.

Truth conditions

- But, we can't just ask our native speaker, what does "snow is white" *mean*? We're bound to get a vague or unhelpful answer. Try this out yourselves (if you want to annoy your friends).
- Instead, we can ask our native speaker, if in a given situation, a sentence of L is *true*, and through asking these questions, we can infer the sentences truth-conditions.

(5) If snow is black, is "snow is white" true?

Truth conditions ii

- In many situations, the results seem bleedingly obvious, but it very quickly gets more interesting (and complicated).

(6) At least one person from every country has eaten most of its national dishes.

- Semantics then, involves developing a recursive procedure for mapping sentences of L to their *truth-conditions*.
- One of the reasons why the Tarskian conjecture doesn't seem so interesting at first sight, is that the language that we're using to describe truth-conditions is *English*.

The meta- and object- language

- We call the language we use to specify the truth-conditions of a given sentence our *metalanguage*.
- We call the language we're interested in interpreting - in this case English - the *object language*.
- When the metalanguage and object language are the same, statements of truth-conditions have the feeling of circularity.
- What we need, if we're going to develop a satisfactory semantic theory, is a completely unambiguous metalanguage, the meaning of which we can state in a completely rigorous way...what we need, if a *formal* language.

Natural language as a formal language

I reject the contention that an important theoretical difference exists between formal and natural languages' (Montague, 1970)

Figure 3: the philosopher Richard Montague (1930-1971)

Interpretation as a *process*

- The *dynamic turn* in formal semantics: meaning doesn't lie in *truth conditions*, but rather in how a given sentence changes the beliefs of the participants (Stalnaker, Heim, Kamp, Groenendijk and Stokhof, etc.).

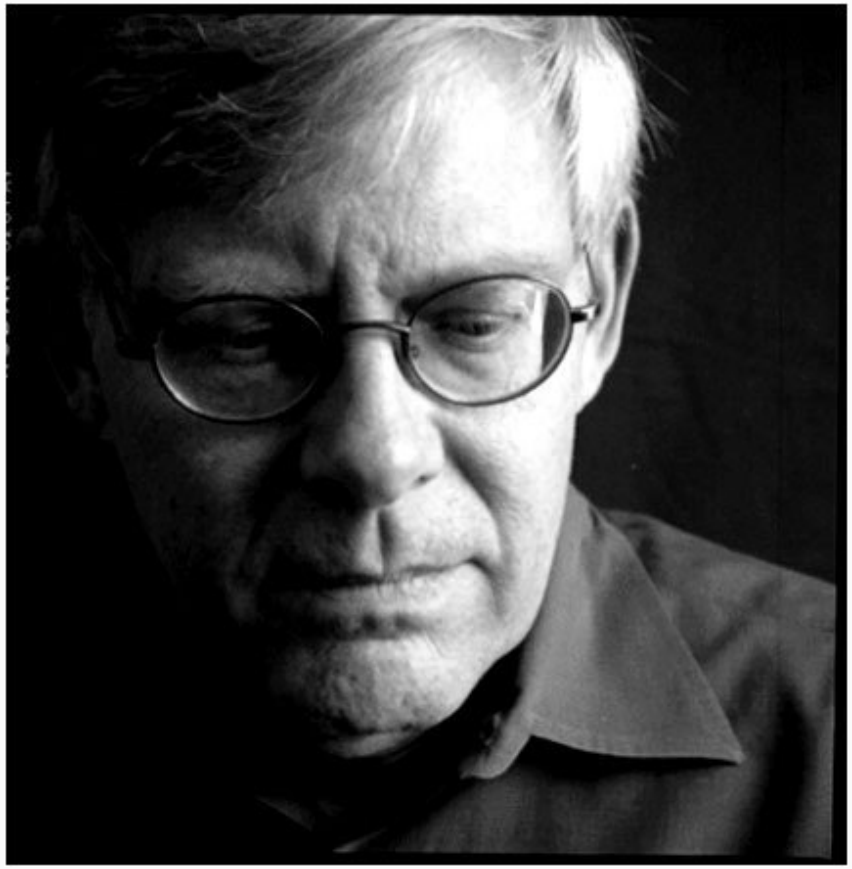


Figure 4: the philosopher Robert Stalnaker

the semantics of programming languages

- The perspective on semantics we'll be working towards has important connections with the semantics of *programming languages* (here we circle back to the Montagovian program).
- In computer science, the *meaning* of a program is how it affects an abstract machine *state*. The *state* can be, e.g., the allocation of values to locations in memory.
- The meaning of a program, therefore, can be thought of as an *instruction for changing the current state*.
- We'll see examples of this in the next session.

Reference

- See Elizabeth Coppock's primer.
- http://eecoppock.info/DynamicSemantics/Lectures/logic-2up.pdf
- If I go too fast for you today, please go back and work through this. The majority of what I'll go through is there, in a condensed format.

Some background, lightning fast

Truth-tables for propositional logic – *and*

- *and* ($\wedge$)

P	Q	$P \wedge Q$
1	1	1
1	0	0
0	1	0
0	0	0

or

- *or (inclusive disjunction)* ($\vee$)

P	Q	$P \vee Q$
1	1	1
1	0	1
0	1	1
0	0	0

if...then...

- *if then (material implication)* ($\rightarrow$)

P	Q	$P \rightarrow Q$
1	1	1
1	0	0
0	1	1
0	0	1

if and only if

- *iff (biconditional)* ($\leftrightarrow$)

P	Q	$P \leftrightarrow Q$
1	1	1
1	0	0
0	1	0
0	0	1

Set theory

This is a set:

$$\{ 2, 4, 6 \}$$

This is a set:

$$\{ x \mid x \text{ is a positive even integer less than } 7 \}$$

These are in fact...the same set.

The empty set

- This is a set:

$$\{\ \}$$

- Another way of writing this set is: $\varnothing$.
- There's only ONE $\varnothing$!!! These are all identical to $\varnothing$:

$$\{\, x \mid x \text{ is a married bachelor} \,\}$$

$$\{\, x \mid x \text{ is an even prime number and } x \text{ is not } 2 \,\}$$

$$\{\, x \mid x \text{ is a reasonably priced flat in Bloomsbury} \,\}$$

Relations on sets

- $A \subseteq B$ – A is a subset of B iff every member of A is a member of B. this means that $\emptyset$ is a subset of every set (including itself!)
- $A \subset B$ – A is a proper subset of B iff $A \subseteq B$ but $A \neq B$.
- $A \cup B$ – the union of A and B is the set of all x s.t. $x \in A$ or $x \in B$.
- $A \cap B$ – the intersection of A and B is the set of all x in both A and B.
- $A - B$ – the difference of A and B is the set of x, s.t. $x \in A$ but $x \notin B$.

Ordered pairs

- $\langle \textbf{Jeff}, \textbf{Britta} \rangle$ is the ordered pair of $\textbf{Jeff}$ and $\textbf{Britta}$.
- unlike sets, ordered pairs are order-sensitive:

  $\langle \textbf{Jeff}, \textbf{Britta} \rangle \neq \langle \textbf{Jeff}, \textbf{Britta} \rangle$
- ordered pairs are objects just like anything else, so we can gather them up in sets.

$$\{ \langle x, y \rangle \mid x \text{ hugs } y \}$$

Functions

- A function is something that takes an *input*, and returns a unique *output*.
- For example the operation f, which takes an integer and adds 1 to it. Here are some different ways of writing this function:

$$f(x) = x + 1$$

$$\begin{bmatrix} 0 \to 1 \\ 1 \to 2 \\ 3 \to 4 \\ \dots \end{bmatrix}$$

Graphs of functions

- f can also be represented as a *set of ordered pairs*, where the first member of the pair is the input, and the second is the output. This is called the *graph* of the function.

$$\{ \ldots, \langle 0, 1 \rangle, \langle 1, 2 \rangle, \langle 3, 4 \rangle, \ldots \}$$

$$\{ \langle x, y \rangle \mid x \text{ is an integer and } y = x + 1 \}$$

Anonymous functions with lambdas

- When we wrote the function f, we had to give it a name...f.
- There's another way to write f without naming it, using a *lambda expression*.

$$\lambda x \,.\, x + 1$$

- A lambda expression consists of a *head* (the bit before the dot), and the *body* (the bit after the dot).
- The function head consists of a λ and an accompanying variable, which tells us where the argument slots into the function body.
- The function body tells us what to do with our argument.

Application

- We *apply* lambda expressions to values like so:

$$[\lambda x \,.\, x + 1](4)$$

- When we apply a lambda expression to an argument, we delete the function head, and substitute all matching occurrences of the variable with the argument, so:

$$= 4 + 1$$

$$= 5$$

Nested lambda expressions

- What if we have a function *subtract* that takes two arguments?

$$\mathit{subtract}(x, y) = y - x$$

- We can write this as a *curried* (nested) lambda expression, like so:

$$\lambda x \,.\, \lambda y \,.\, y - x$$

Nested lambda expressions ii

- Application is left-associative:

$$
\begin{aligned}
&[\lambda x \,.\, \lambda y \,.\, y - x](4)(5) \\
&= [\lambda y \,.\, y - 4](5) \\
&= 5 - 4 \\
&= 1
\end{aligned}
$$

- Note that out the output of applying our function to 4 is *itself* a function: $\lambda y \,.\, y - 4$.
- Lambda expressions come from the *lambda calculus* – a formal system for expressing computation. It's something of a lingua franca in formal semantics, but it's a big topic, and I'll try to avoid using it in this class.

Our first formal language: L_1

First-order logic without variables

- We'll define a language for dealing with things like verbs and proper names. Let's call it L_1.
- We'll be able to use this language for specifying the truth-conditions of sentences like the following:

  (7) If Jeff is annoying, then Britta and Annie are bored.
  $$(\text{annoying}(J)) \rightarrow ((\text{bored}(B) \wedge \text{bored}(A)))$$

  (8) Troy likes Abed and not Pierce.
  $$\text{likes}(T, A) \wedge \neg\text{likes}(T, P)$$

- Don't worry about how to interpret these logical translations just yet. We'll get onto that in a sec.

Logical Form

- The logical translation (and hence disambiguation) of natural language sentences corresponds to philosophers' notion of *Logical Form*.

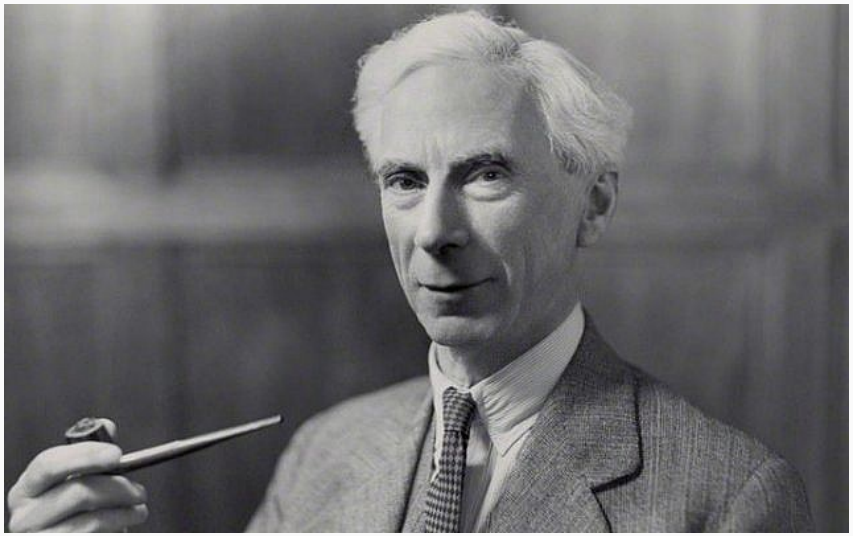


Figure 5: the philosopher Bertrand Russell (1872-1970)

- This is an important idea philosophically, as the inferential properties of a sentence can be read directly off of its Logical Form.

Describing a formal language

- Just like Montague said, the tasks of describing a formal language like L_1 and a natural language like English are really quite similar.
- Just like English, L_1 has a *vocabulary*, which is exactly what it sounds like.
- Just like English, L_1 has a *syntax* – this will consist of a set of rules for constructing grammatical sentences of L_1, which, following the parlance of logicians, we'll call *well-formed formulae* (wff) of L_1.
- Because L_1 abstracts away from many of the complexities of English, including ambiguity, we'll also be able to give a *semantics* for L_1, which will consist of a procedure for computing the truth-conditions of any given wff.

Vocabulary of L_1

1. Individuals contstants/terms: Troy, Annie, Britta, etc

   These correspond to *proper names* and *definite descriptions* in English.

2. Unary predicates: happy, bored, etc

   These correspond to *adjectives* and *intransitive verbs* in English.

3. Binary predicates: kissed, loves

   These correspond to *transitive verbs* in English.

Syntax of L_1

1. If π is a unary predicate and α is a term, then $\pi(\alpha)$ is a formula.
2. If π is a binary predicate and α and β are terms, then $\pi(\alpha, \beta)$ is a formula.
3. If ϕ is a formula, then $\neg\phi$ is a formula.
4. If ϕ and ψ are formulas, then $[\phi \wedge \psi]$ is a formula.
5. If ϕ and ψ are formulas, then $[\phi \vee \psi]$ is a formula.
6. If ϕ and ψ are formulas, then $[\phi \rightarrow \psi]$ is a formula.
7. If ϕ and ψ$ are formulas, then $[\phi \leftrightarrow \psi]$ is a formula.

An aside on recursion

Figure 6: Morris Halle (1923-2018) and Noam Chomsky (src: Kai von Fintel)

recursion ii

For example, the most elementary property of the language faculty is the property of discrete infinity; you have six-word sentences, seven-word sentences but you don't have six-and-a-half-word sentences. Furthermore, there is no limit; you can have ten-word sentences, twenty-word sentences and so on indefinitely. That is the property of discrete infinity. (Chomsky, 2000 – The Architecture of Language*)*

Chomskyan rewrite rules

- Task: use the following set of rewrite rules to generate some sentences. Convince yourself that there is no limit to the number of sentences you can generate.

$$V \rightarrow \text{hugs} \mid \text{kisses}$$

$$V' \rightarrow \text{say} \mid \text{believe}$$

$$NP \rightarrow \text{Britta} \mid \text{Annie}$$

$$VP \rightarrow V\ NP$$

$$VP \rightarrow V'\ S$$

$$S \rightarrow NP\ VP$$

- What is the crucial component here that is responsible for your ability to generate an unlimited number of different sentences?

An even simpler instance of recursion

$$S \rightarrow S \text{ and } S$$

$$S \rightarrow S \text{ or } S$$

$$S \rightarrow \text{if } S \text{ then } S$$

- Does this look familiar?

Discrete infinity

- our specification of the syntax of L_1 is *recursive*, just like Chomsky's rewrite rules!

4. If ϕ and ψ are formulas, then $[\phi \wedge \psi]$ is a formula.
5. If ϕ and ψ are formulas, then $[\phi \vee \psi]$ is a formula.
6. If ϕ and ψ are formulas, then $[\phi \rightarrow \psi]$ is a formula.

- If we're gonna have *any* hope of analysing the meaning of a natural language using a *formal language* such as L_1, we better hope that our formal language is capable of matching natural language's capacity for discrete infinity.

Questions on syntax of L_1

- Given a suitable vocabulary, which of the following are formulae of L_1?

1. $\neg\neg\text{happy}(M)$
2. $\text{happy}(A)$
3. $\text{happy}(M, J)$
4. $\text{loves}(J, M) \vee \text{loves}(M, M)$
5. $[\text{loves}(J, M) \leftrightarrow \text{loves}(M, J)]$
6. $[\text{kissed}(M) \leftrightarrow \text{loves}(M, J)]$
7. $\neg[\text{loves}(J, M) \leftarrow \neg\text{loves}(M, J)]$

Semantics

- The semantics for L_1 is given in terms of a *model*, i.e., a mathematical description of a toy universe.

Figure 7: src: *A Philosopher Lecturing on the Orrery*, Joseph Wright of Derby

Semantics ii

$$M = \langle D, I \rangle$$

- D is the *domain* of all individuals in M.
- I is the *interpretation function*; it tells us what the vocabulary of L_1 means in M.
    - our mathematical description of the toy universe therefore, tells us who and what exists, and what the expressions in the vocabulary of the language we're interested in pick out.

Semantics iii

- The interpretation of an *individual constant*, which you can think of as a proper name, will be an individual in the domain.
- The interpretation of a unary predicate will be a *set* of individuals.

  e.g., the interpretation of *run* is the set of individuals who run.
- The interpretation of a binary predicate will be a binary relation between individuals, i.e., a set of ordered pairs.

  e.g., the interpretation of *hugs* is the set of ordered pairs $\langle x, y \rangle$, such that x hugs y.

A note on type-setting conventions

- When we think though the meaning of the English NP “Homer”, we need to distinguish three levels:
    - The object-language expression “Homer”, which will be set in the same typeface as my notes. This is an NP/DP in the natural language English.
    - The metalanguage expression Homer, which is an individual constant in the vocabulary of our formal language – I’ll set metalanguage expressions in a serif typeface.
    - The person `Homer`, who is an entity in our domain. I’ll typeset entities in the domain in a fixed-width typeface.
- It’s important to be mindful of these distinctions as we move forwards...I’m not perfect, and I might typeset things incorrectly sometimes. It’s good practice for you to correct me if/when this happens!

Semantics iv

- task: Add some *ternary predicates* to L_1 (i.e., predicates that take three arguments).
  - what is the interpretation of a ternary predicate in $\boldsymbol{M}$?
  - what syntactic category does a ternary predicate correspond to, intuitively?

Semantics of L_1 (definition)

- If α is a predicate or an individual constant, then:
  - $[\![\alpha]\!]^M = I(\alpha)$.
- If π is a unary predicate and α is a individual constant, then:
  - $[\![\phi(\alpha)]\!]^M = 1$ if $[\![\alpha]\!]^M \in [\![\pi]\!]^M$
  - $[\![\phi(\alpha)]\!]^M = 0$ otherwise
- N.b. we're gonna save ourselves some time in the future by writing *if and only if* (iff), to give truth and falsity conditions in one line. Since we're assuming the excluded middle (every formula is either true or false), this is harmless.
- Note that $[\![.]\!]^M$ is itself a *function* from any vocabulary item or formula α of L_1 to the semantic value of α in our model M.

Semantics of L_1 continued

- if π is a binary predicate, and α and β are terms, then:
  - $[\![\pi(\alpha, \beta)]\!]^M = 1$ iff $\langle [\![\alpha]\!]^M, [\![\beta]\!]^M \rangle \in [\![\pi]\!]^M$
- $[\![\neg\phi]\!]^M = 1$ iff $[\![\phi]\!]^M = 0$
- $[\![\phi \wedge \psi]\!]^M = 1$ iff $[\![\phi]\!]^M = 1$ and $[\![\psi]\!]^M = 1$
- $[\![\phi \vee \psi]\!]^M = 1$ iff $[\![\phi]\!]^M = 1$ or $[\![\psi]\!]^M = 1$
- $[\![\phi \rightarrow \psi]\!]^M = 1$ unless $[\![\phi]\!]^M = 1$ and $[\![\psi]\!]^M = 0$
- $[\![\phi \leftrightarrow \psi]\!]^M = 1$ iff $[\![\phi]\!]^M = [\![\psi]\!]^M$

Recursion

- Let's look again at our rule for the semantic value of a conjunctive formula:
- $[\![\phi \wedge \psi]\!]^M = 1$ iff $[\![\phi]\!]^M = 1$ and $[\![\psi]\!]^M$
- $[\![.]\!]$...is a recursive function! In other words, it can recycle it's own output as input.
- If we have a recursive syntax, then we better have a recursive semantics.

Recursive semantics

- Compute the truth-conditions of the following well-formed formula of L_1.

$$(\text{happy}(\text{Jeff}) \wedge \text{happy}(\text{Troy})) \wedge \text{bored}(\text{Britta})$$

Exercise

- Convert the following sentences of English into well-formed formulae of L_1, and compute their truth-conditions.

(9) If Jeff is happy, then Britta and Annie are bored.

(10) if Jeff is happy, then Britta is bored or Annie is bored.

(11) Britta isn't bored or happy.

- Make a note of any interesting decision points in translating the sentences of English into formulae of L_1.

English as a formal language

- have we made good on Montague's conjecture?
- Right now, we can only deal with proper names and verbs – we still have a long way to go.
- AND as you may have noticed, the procedure for translating English into L_1 relies to a large extent on intuition.

(12) If Jeff is happy, then Britta is bored or Annie is bored.

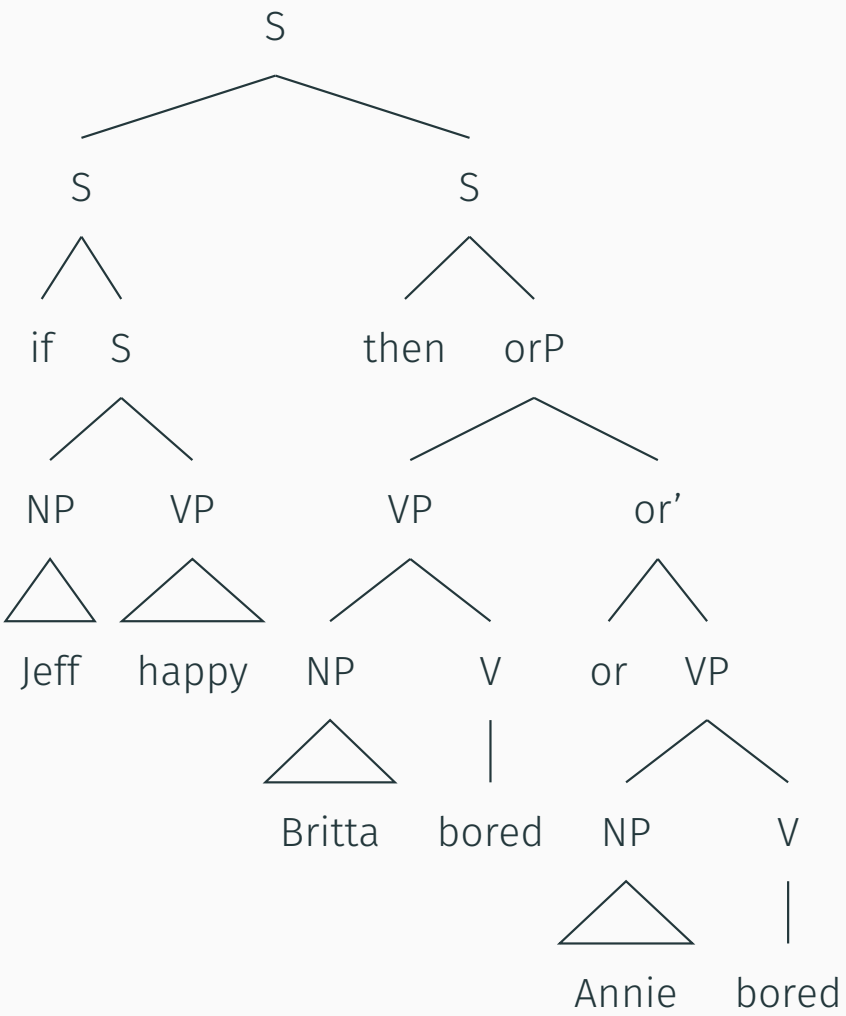
- There were two valid translations:

(13) $\text{happy}(J) \rightarrow (\text{bored}(B) \vee \text{bored}(A))$

(14) $(\text{happy}(J) \rightarrow \text{bored}(B)) \vee \text{bored}(A)$

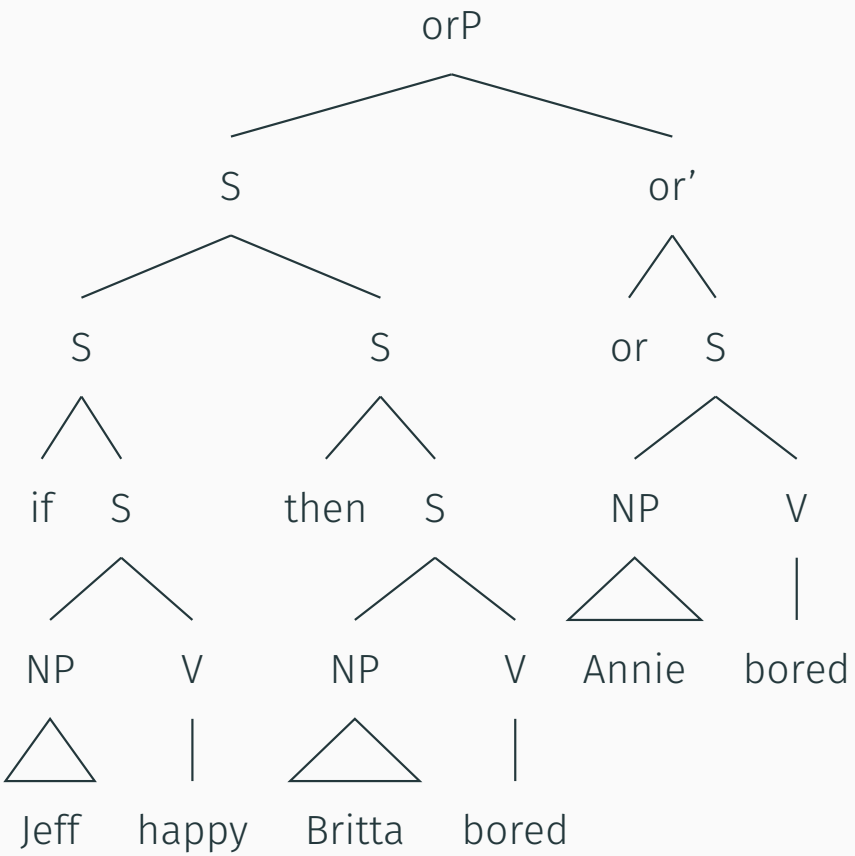
- Can our procedure for translating sentences of English into formulae ever be completely deterministic? What would help?

Structural ambiguity



(15) $\text{happy}(J) \rightarrow (\text{bored}(B) \vee \text{bored}(A))$

Structural ambiguity ii



(16) $(\text{happy}(J) \rightarrow \text{bored}(B)) \vee \text{bored}(A)$

Compositional semantics

- What we want is a procedure for mapping each *terminal node* to an element of our *vocabulary*, and each *non-terminal node* to a well-formed formula.
- This is the study of *syntax-semantics interface*, and encompasses a great deal of what formal semanticists do, especially in the generative tradition.
- I won't really be going into much depth about how this works in this class, but if you're interested, the *locus classicus* is Heim & Kratzer (1998), but see also Jacobson (2014) for an excellent contemporary introduction.

Compositional semantics ii

- Instead, we're going to rely on our semantic intuitions for translating English into a formal language, and go from there.
- All of the action will be in thinking through what *syntactic* properties our formal language should have, and moreover, specifying a semantics for the language which indirectly captures our intuitions about the truth-conditions of English sentences.

- Consider the following sentences of English:

(17) Someone is happy.

(18) It's not the case that everyone is bored.

(19) She is sitting down.

- Rather than using proper names, we're using *quantificational NPs* like *someone*, and *pronouns* like *she*.

Syntax of FOL

- Note that quantificational NPs and pronouns have the same distribution as proper names.
- This makes it easy enough to add some extra resources to L_1.

Syntax of FOL ii

- For pronouns we're going to add *variables*, for which we'll use the letters x, y, z, etc. We'll refer to both *individual constants* like Jeff and variables as *terms*.

  Why not just add *he, she, they* etc., to our vocabulary as constants? We'll see why when we get to the semantics.

- We're also going to add two new logical operators: $\exists$, and $\forall$, which correspond to *someone* and *everyone* (roughly).

Syntax of FOL iii

- the first part of the syntax is just our syntax of L_1, but with *individual constant* replaced by *term*. This means that variables can occupy the same environments as proper names.

1. If π is a unary predicate and α is a term, then $\pi(\alpha)$ is a formula.
2. If π is a binary predicate and α and β are terms, then $\pi(\alpha, \beta)$ is a formula.
3. If ϕ is a formula, then $\neg\phi$ is a formula.
4. If ϕ and ψ are formulas, then $[\phi \wedge \psi]$ is a formula.
5. If ϕ and ψ are formulas, then $[\phi \vee \psi]$ is a formula.
6. If ϕ and ψ are formulas, then $[\phi \rightarrow \psi]$ is a formula.
7. If ϕ and ψ$ are formulas, then $[\phi \leftrightarrow \psi]$ is a formula.

Syntax of FOL iv

- Quantificational operators are accompanied by variables:

8. If u is a variable and ϕ is a formula, then $\exists u \phi$ is a formula.

9. If u is a variable and ϕ is a formula, then $\forall u \phi$ is a formula.

Open vs. closed formulae

- A formula is *open* if it contains free variables, and *closed* otherwise.
- Free variables are those that are not *bound*.
- A variable is *bound* by the closest matching quantifier that takes scope over it.
- Task: for each of the following formulae, is it open or closed:

1. $\exists y[\text{likes}(x, y)]$
2. $\forall x[\text{happy}(x)] \wedge \text{arrived}(x)$
3. $\exists z[\text{hugs}(z)(z)]$

Towards a semantics of FOL

- Before we get into the formal nitty-gritty, I want you to think about the meaning of the following sentence. Is it possible to express in terms of *truth-conditions*?

  (20) She is Donald Trump's wife.

- What factors does the interpretation of (20) depend on?

Assignments

- Intuitively, whether or not (20) is true, depends on who the speaker intended to refer to with the pronoun *she*.
- If the speaker intended to refer to Melania Trump, then (20) is true, and otherwise it is false.

Assignments ii

- We can use multiple occurrences of the same pronoun to refer to different individuals.

  (21) She is Donald Trump's wife and *she* is Bill Clinton's wife.

- For this reason, we have to relativize the truth of a sentence to who the speaker intended each *tokening* of a pronoun to refer to.

- This is why we need something like variables in our formal language. We need a device for distinguishing between occurrences of pronouns.

Assignments iii

- As a notational convenience, we will write things like this:

(22) She$_x$ is Donald Trump's wife, and she$_y$ is Bill Clinton's wife.

- This tells us that the first *she* corresponds to the variable x in the logical translation, and the second *she* corresponds to the variable y.

Assignments iv

- Ok, so the translation of *She*$_x$ *is Donald Trump's wife* is the following:

$$\text{wife}(\text{Trump}, x)$$

- How do we capture the notion that the truth of this formula is relative to who the pronoun refers to?
- We're going to *relativise* truth, to a *context* $\mathbf{g}$. Formally, $\mathbf{g}$ is a function from variables to objects, i.e., an *assignment function*.

Assignments v

$$g_1 = \begin{bmatrix} x & \mapsto \texttt{Melania} \\ y & \mapsto \texttt{Hilary} \\ \dots & \end{bmatrix}$$

$$g_2 = \begin{bmatrix} x & \mapsto \texttt{Hilary} \\ y & \mapsto \texttt{Melania} \\ \dots & \end{bmatrix}$$

Assignments vi

- $\mathbf{wife}(\mathbf{Trump}, x)$ is true relative to the assignment $\mathbf{g_1}$ but false relative to the assignment $\mathbf{g_2}$.

Assignments vii

- From now on, interpretation isn't just relative to a model M, but also relative to an assignment g.
- Instead of $[\![\phi]\!]^M$, we write $[\![\phi]\!]^{M,g}$.
- The interpretation of a variable x is just whatever the assignment g maps it to.
- $[\![x]\!]^{M,g} = g(x)$
- $[\![x]\!]^{M,g_1} = \texttt{Melania}$; $[\![x]\!]^{M,g_2} = \texttt{Hilary}$

Semantics of FOL

- The most important new rule we'll be dealing with is the following:
- If α is a variable, then $[\![\alpha]\!]^{M,g} = g(\alpha)$
- Most of the rest of our semantic rules will be identical to those of L_1, but with an additional parameter for the assignment g.

Semantics of FOL ii

- If α is a predicate or an individual constant, then:
  - $[\![\alpha]\!]^{M,g} = I(\alpha)$.
- If π is a unary predicate and α is a individual constant, then:
  - $[\![\phi(\alpha)]\!]^{M,g} = 1$ iff $[\![\alpha]\!]^{M,g} \in [\![\pi]\!]^{M,g}$
- if π is a binary predicate, and α and β are terms, then:
  - $[\![\pi(\alpha, \beta)]\!]^{M,g} = 1$ iff $\langle [\![\alpha]\!]^{M,g}, [\![\beta]\!]^{M,g} \rangle \in [\![\pi]\!]^{M,g}$

Semantics of FOL iii

- $[\![\neg\phi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 0$
- $[\![\phi \wedge \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 1$ and $[\![\psi]\!]^{M,g} = 1$
- $[\![\phi \vee \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 1$ or $[\![\psi]\!]^{M,g} = 1$
- $[\![\phi \rightarrow \psi]\!]^{M,g} = 1$ unless $[\![\phi]\!]^{M,g} = 1$ and $[\![\psi]\!]^{M,g} = 0$
- $[\![\phi \leftrightarrow \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = [\![\psi]\!]^{M,g}$

Pronouns

- We now have all of the resources we need for translating a subset of sentences of English with pronouns into FOL, and assigning them truth-conditions. Do this for the following sentence:

  (23) She arrived and she sat down.

- Make a note of the decisions you have to make when you translate (23) into FOL.

  - What do you notice?
  - How does it affect the truth conditions.

Pronouns ii

- There are two candidate translations (Logical Forms) for (23):

(24) $\text{arrived}(x) \wedge \text{satDown}(x)$

(25) $\text{arrived}(x) \wedge \text{satDown}(y)$

Co-indexation

$$[\![\text{arrived}(x) \wedge \text{satDown}(x)]\!]^{M,g} = 1 \text{ iff}$$

$$g(x) \in I(\text{arrived}) \text{ and } g(x) \in I(\text{satDown})$$

- Since an *assignment function* is a *function*, when we map two pronouns to the same variable, they are *guaranteed* to pick out the same individual relative to any assignment.

Contra-indexation

$$[\![\text{arrived}(x) \wedge \text{satDown}(y)]\!]^{M,g} = 1 \text{ iff}$$

$$g(x) \in I(\text{arrived}) \text{ and } g(y) \in I(\text{satDown})$$

- when we map two pronouns to distinct variables, is it *guaranteed* that they pick out distinct variables, relative to any assignment?

Contra-indexation ii

- no.
- Consider the assignment $g_c = \begin{bmatrix} x & \mapsto \texttt{Hilary} \\ y & \mapsto \texttt{Hilary} \end{bmatrix}$

$$[\![\text{arrived}(x) \wedge \text{satDown}(y)]\!]^{M,g_c} = 1 \text{ iff}$$

$$\texttt{Hilary} \in I(\text{arrived}) \text{ and } \texttt{Hilary} \in I(\text{satDown})$$

Contra-indexation iii

- but, if we map two pronouns to distinct variables, we make it *possible* for them to pick out distinct individuals. Consider the following assignment:

$$g_d = \begin{bmatrix} x & \mapsto \texttt{Hilary} \\ y & \mapsto \texttt{Melania} \end{bmatrix}$$

$$[\![\text{arrived}(x) \wedge \text{satDown}(y)]\!]^{M,g_d} = 1 \text{ iff}$$

$\texttt{Hilary} \in I(\text{arrived})$ and $\texttt{Melania} \in I(\text{satDown})$

Assignments and contexts

- We can think of an assignment g as formalising aspects of the *context of utterance*.
- g_c represents a context in which both "she$_x$" and "she$_y$" are intended to pick out `Hilary`.
- g_d represents a context in which "she$_x$" is intended to pick out `Hilary`, and "she$_y$" is intended to pick out `Melania`.

Assignments and the global environment

- Computer programs often need access to a *global environment*.
- For example, imagine a program that pull's a user's first name and surname from a database, and concatenates them.
- The *meaning* of this program can be thought of as assignment sensitive:

```
query g
println (g(firstName) ++ g(surname))
```

- g here is an assignment function, standing in for an entry in a database specifying an individuals first name and surname.

Assignments and the global environment ii

- The formal tools computer programmers use for thinking about the role the global environment plays look *extremely* similar to the tools we're using to analyse the *context*.
- If you have some background in programming, take a look at the *Reader* monad in functional languages like *Haskell*.
- If not don't worry - hopefully the basic idea is intuitive enough.

Back to quantificational NPs

- we still haven't said anything about the meaning of sentences like the following:

  (26) Someone arrived.

  (27) Everyone arrived.

- It turns out that *assignments* provide us with the machinery to analyse these cases too.

The informal idea

(28) Someone arrived.

(29) $[\![\exists x[\text{arrived}(x)]]\!]^{M,g}$

- When is (29) true relative to an assignment g?
- Intuitively, it's truth is *assignment-invariant* – that is to say that it doesn't depend on what the context of utterance is.
- It's true just in case we can find *any* assignment g', which is identical to g except for what it assigns x to, which makes (29) true.

Illustration

(30) $\texttt{arrive} = \{\,\texttt{Melania}\,\}$

(31) $g = \begin{bmatrix} x & \mapsto \texttt{Hilary} \\ y & \mapsto \texttt{Melania} \end{bmatrix}$

- the truth of $[\![\exists x[\text{arrived}(x)]]\!]^{M,g}$ doesn't depend on what g maps x to, but rather if we can find a g' that maps x to someone that arrived.
- We can find such a g': $\begin{bmatrix} x & \mapsto \texttt{Melania} \\ y & \mapsto \texttt{Melania} \end{bmatrix}$

Formalism

- We can state the rule for quantificational sentences more formally like so;
- $[\![\exists v\ \phi]\!]^{M,g} = 1$ iff there is at least one g' s.t. $g'[v]g$ and $[\![\phi]\!]^{M,g'} = 1$
- $[\![\forall v\ \phi]\!]^{M,g} = 1$ iff for every g' s.t. $g'[v]g$, $[\![\phi]\!]^{M,g'} = 1$

Minimally differing assignments

- Why do we look at only *minimally* differing assignments?

(32) $\exists x[\text{arrive}(x) \wedge \text{satDown}(y)]$

- Let's say that we're in a context with the following g: $\begin{bmatrix} x & \mapsto \texttt{Melania} \\ y & \mapsto \texttt{Hilary} \end{bmatrix}$
- And let's say that it's true that Hilary arrived but only Melania sat down.

Minimally differing assignments ii

- Well, if we're allowed to look at assignments that differ in more than just what x gets mapped to, then we *can* find an assignment g' which makes the formula true.

$$g' = \begin{bmatrix} x & \mapsto \texttt{Hilary} \\ y & \mapsto \texttt{Melania} \end{bmatrix}$$

- This is clearly a bad prediction though – the truth of the previous formula should depend on who y picks out in the context of utterance.

Non-determinism

- We can think of existential quantifiers as triggering a *non-deterministic computation*. When considering the truth of $\exists x[\text{arrive}(x)]$, we compute the truth of statements of the form $\alpha \in I(\text{arrive})$, where α is some object.
- As soon as we find one that is true, we travel back to a deterministic world, by stating that the formula is true.

Non-determinism ii

- This is going to be important once we start looking at dynamic semantics.
- For now it is important to remember that this is just a metaphor to help you think about what existential quantification is doing. See Simon Charlow's dissertation for some (very advanced) reading on the connections between existential quantification and non-determinism.

Binding

- The fact that quantificational operators manipulate the assignment function can be used to analyse the fact that quantifiers can *bind* pronouns, reflexives, and other anaphora in natural language.

  (33) Someone^{x} likes themselves$_{x}$.

  (34) $\exists x[\text{likes}(x, x)]$

- Binding obtains just in case the variable introduced by the quantificational operator matches the variable introduced by the pronoun.

Interpreting binding

- $[\![\exists x[\text{likes}(x, x)]]\!]^{M,g} = 1$ iff...
- $\exists g'[g[x]g'$ and $[\![\text{likes}(x, x)]]\!]^{M,g'} = 1$
- $\exists g'[g[x]g'$ and $\langle g'(x), g'(x)\rangle \in I(\text{likes})$
- This is true, if we're in a model, e.g., where..

$$I(\text{likes}) = \{\, \langle \text{Melania}, \text{Melania}\rangle, \langle \text{Melania}, \text{Hilary}\rangle \,\}$$

- and an assignment g, s.t. $g(x) = \text{Hilary}$
- Since we can find an assignment $g[x]g'$ which makes the embedded formulae true. Namely g' s.t., $g'(x) = \text{Melania}$.

A note on crossover

- The classical conception of binding gives rise to the *crossover* problem.
- The following is an example of *strong crossover*.

(35) She$_x$ likes someone^x.

- Why can we not translate this as: $\exists x[\text{likes}(x, x)]$

Exercises

- Translate the following sentences of English into FOL and compute their truth-conditions. To make life easier for you, I'm going to diambiguate indexation.

  (36) It's not the case that someone^{x} likes themselves$_{x}$.

  (37) If someone is upset, then everyone is unhappy.

  (38) Someone danced with everyone.

- Note all of these sentences are ambiguous. Pick a reading when you translate. Think about how FOL represents the ambiguity.

Towards a semantics of discourses

- Translate the following into FOL:

(39) Someone walked in. She$_x$ sat down.

- You can assume that two consectutive sentences are *conjoined*.

Candidate translations

- Conjunction scopes over existential:

(40) $[\exists x[\text{walkedIn}(x)]] \wedge \text{satDown}(x)$

- Existential scopes over conjunction:

(41) $\exists x[\text{walkedIn}(x) \wedge \text{satDown}(x)]$

- *Exercise:* Compute the truth conditions of both formulae. What do you notice?

Conjunction scopes over existential

(42) $[\![\exists x[\text{walkedIn}(x)] \wedge \text{satDown}(x)]\!]^{M,g} = 1$ iff

$\exists g'[g[x]g' \wedge g'(x) \in I(\text{walkedIn})] \wedge (g(x) \in I(\text{satDown}))$

- This seems like a *bad* translation. It can be true, e.g., if only Hilary walked in, and only Melania sat down.
- Intuitively, it doesn't matter than the two variables are the same, since the second variable is outside of the scope of the existential quantifier.

Existential scopes over conjunction

(43) $[\![\exists x[\text{walkedIn}(x) \wedge \text{satDown}(x)]]\!]^{M,g} = 1$ iff

$\exists g'[g[x]g' \wedge g'(x) \in I(\text{walkedIn}) \wedge g'(x) \in I(\text{satDown})]$

- This seems like a good translation – it is only true the same person both walked in and sat down. This is because both variables are within the scope of the existential quantifier.

Whats's the problem

- Well, why don't we just allow existential quantifiers to scope out of the sentences which contain them?
- Problem 1: other quantificational NP's don't behave like this.

(44) Everyone walked in. He_x sat down.

This cannot be translated as:

(45) $\forall x[\text{walkedIn}(x) \rightarrow \text{satDown}(x)]$

What's the problem ii

- The most robust methodological principle in formal semantics is Frege's *principle of compositionality*.

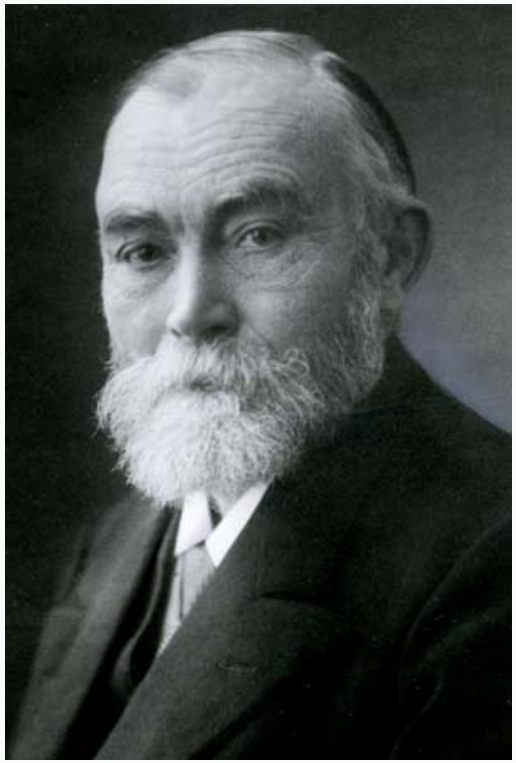


Figure 8: the philosopher Gottlob Frege (1848-1925)

What's the problem iii

- Frege's principle of compostionality has had innumerable reformulations, and I won't bore you with the original German. In its essence, it says:

  The meaning of a sentence is a function of the meaning of its parts and how they are put together.

- Let's think about how we would translate the *parts* of the following sentence:

(46) Someone walked in and he sat down.

What's the problem iv

- conjunct 1: $\exists x[\text{walkedIn}(x)]$
- conjunct 2: $\text{satDown}(x)$
- The principle of compositionality tells us that the meaning of a *conjunctive* sentence should be the conjunction of the meaning of its parts, so we get:
- $\exists x[\text{walkedIn}(x)] \wedge \text{SatDown}(x)$
- our more successful translation violates the principle of compositionality!

What's the problem v

- It's important to state that, in practice, the principle of compositionality is a *methodological* principle, and is not always inviolable.
- However, all else being equal, it would be nice to be able to account for the data we've just been considering in a way consistent with the principle of compositionality.
- *Dynamic semantics* will allow us to do just that!

Lecture 2: Dynamic Predicate Logic

The empirical challenge

"If we use standard first-order predicate logic [...] in translating a natural language sentence or discourse, anaphoric pronouns will turn up as bound variables. In many cases, this means that in order to arrive at formulas which are good translations, i.e., which express the right meaning, we have to be pretty inventive, and should not pay too much attention to the way in which the natural language sentence or discourse is built up." (Groenendijk & Stokhof 1991)

Cross-sentential anaphora

(47) A man^{x} walks in the park. He$_{x}$ whistles.

(48) $\exists x[\mathrm{man}(x) \wedge \mathrm{walkInThePark}(x) \wedge \mathrm{whistle}(x)]$

- BUT

(49) A man walks in the park – $\exists x[\mathrm{man}(x) \wedge \mathrm{walkedInThePark}(x)]$

(50) He$_{x}$ whistles. – $\mathrm{whistles}(x)$

- (51) $\neq$ (49) $\wedge$ (50)

Donkey sentences

(52) If a farmer owns a donkey^{y}, he$_{x}$ beats it$_{y}$.

(53) Every farmer who owns a donkey^{y} beats it$_{y}$.

$$\forall x \forall y \left[\begin{array}{l} [\text{farmer}(x) \wedge \text{donkey}(y) \wedge \text{own}(x, y)] \\ \rightarrow \text{beat}(x, y) \end{array} \right]$$

- what happened to translating indefinites like *a donkey* using existential quantification?

The DPL answer

"The general starting point of the kind of semantics that 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" (Groenendijk & Stokhof 1991)

The syntax of DPL

- The syntax of DPL is more-or-less identical to the syntax of FOL!
- Let's have a brief refresher.

The syntax of DPL ii

1. If π is a unary predicate and α is a term, then $\pi(\alpha)$ is a formula.
2. If π is a binary predicate and α and β are terms, then $\pi(\alpha, \beta)$ is a formula.
3. If ϕ is a formula, then $\neg\phi$ is a formula.
4. If ϕ and ψ are formulas, then $[\phi \wedge \psi]$ is a formula.
5. If ϕ and ψ are formulas, then $[\phi \vee \psi]$ is a formula.
6. If ϕ and ψ are formulas, then $[\phi \rightarrow \psi]$ is a formula.
7. If ϕ and ψ$ are formulas, then $[\phi \leftrightarrow \psi]$ is a formula.
8. If u is a variable and ϕ is a formula, then $\exists u\phi$ is a formula.
9. If u is a variable and ϕ is a formula, then $\forall u\phi$ is a formula.

The syntax of DPL iii

- The difference between DPL and FOL is going to lie in the *semantics*.
- Concretely, formulae like (54) are going to receive sensible interpretations.

(54) $\exists x[\mathrm{man}(x) \wedge \mathrm{walkedIn}(x)] \wedge \mathrm{satDown}(x)$

- First we need some background on characteristic functions and sets.
- We're also going to develop a slightly different perspective on the meanings of FOL formulae.

Prelude to the semantics of DPL: Characteristic functions

- Let's say we have a function $\texttt{run'}$ from objects to truth values.
- $\texttt{run'}$ maps an object x to true (1) just in case x runs, and to false (0) just in case x doesn't run.
- Let's say we're in a world where Jeff runs, but Britta and Annie don't run.

(55) $\texttt{run'(Jeff)} = 1$

(56) $\texttt{run'(Britta)} = 0$

(57) $\texttt{run'(Annie)} = 0$

Characteristic functions ii

- The graph of `run'` in this world is:

$$\{ \langle \text{Jeff}, 1 \rangle, \langle \text{Britta}, 0 \rangle, \langle \text{Annie}, 0 \rangle \}$$

- We could convey the same information by simply gathering together every object for which `run'` returns true:

$$\{ \text{Jeff} \}$$

- This is the meaning we assume for predicates in FOL. `run'` is the *characteristic function* of this set.
- Functions from a set of objects to truth values can equivalenty be expressed as sets of objects.

Sentences express sets of assignments

- Recall in FOL we considered truth-conditions to be relative to an assignment g.

$$[\![\text{left}(x)]\!]^{M,g} = 1 \text{ iff } g(x) \in \texttt{left}$$

- Let's say that only Donald left. If we have an assignment g_1 s.t. $g_1(x) = \text{Donald}$ then the formula is true. If we have an assignment g_2 s.t. $g_2(x) = \text{Melania}$ then the formula is false.
- In FOL, then, formulae are *functions* from assignments to truth values.
- That means that we can identity the meaning of a formula relative to a model with the set of assignments that make it true.

Sentences express sets of assignments

- Let's say we've in a model where only Jeff and Britta run, and only Britta and Annie swim.

$$[\![\text{run}(x)]\!]^M = \left\{ \begin{array}{ll} [x & \mapsto \texttt{Jeff}], \\ [x & \mapsto \texttt{Britta}] \end{array} \right\}$$

The flow of information

- This allows us to observe the flow of information as we added conjucts especially clearly.

$$[\![\text{run}(x)]\!]^M = \left\{ \begin{array}{ll} [x & \mapsto \texttt{Jeff}], \\ [x & \mapsto \texttt{Britta}] \end{array} \right\}$$

$$[\![\text{swim}(x)]\!]^M = \left\{ \begin{array}{ll} [x & \mapsto \texttt{Britta}], \\ [x & \mapsto \texttt{Annie}] \end{array} \right\}$$

$$[\![\text{run}(x) \wedge \text{swim}(x)]\!]^M = \{\, [x \mapsto \texttt{Britta}] \,\}$$

- Formulae involving pronouns makes the set of assignments at which the sentence is true *shrink*.

Notation for assignments

- I'm going to start using more concise notation for assignment functions now. $j\ b\ j$ is the assignment that maps x_1 to `Jeff`, x_2 to `Britta` and x_3 to `Jeff`.
- The assignment $j\ b\ j$ is equivalent to the following:

$$\begin{bmatrix} x_1 & \mapsto \texttt{Jeff} \\ x_2 & \mapsto \texttt{Britta} \\ x_3 & \mapsto \texttt{Jeff} \end{bmatrix}$$

Notation for assignments ii

- $\left\{ \begin{array}{l} j\ j \\ j\ b \\ j\ a \end{array} \right\}$ is the set of assignments which always map x_1 to `Jeff`.
- Intuitively we can think of this as a context where the value of x_1 is known.

Notation for assignments iii

- This notation will help us see more clearly how, as we add conjuncts with free variables, the level of uncertainty is reduced. Let's say there are two variables, x_1, x_2.

$$[\![\text{run}(x_1)]\!]^M = \left\{ \begin{array}{c} j\ j \\ j\ b \\ j\ a \\ b\ b \\ b\ j \\ b\ a \end{array} \right\};\ [\![\text{swim}(x_1)]\!]^M = \left\{ \begin{array}{c} b\ b \\ b\ j \\ b\ a \\ a\ a \\ a\ b \\ a\ j \end{array} \right\}$$

Notation for assignments ii

$$[\![\text{run}(x_1) \wedge \text{swim}(x_1)]\!]^M = \left\{ \begin{array}{l} b\ b \\ b\ j \\ b\ a \end{array} \right\}$$

- Conjoining the two statements about x_1 means that we can only be in a context where we're completely certain that x_1 picks out `Britta`.

The flow of information ii

- Existential statements, on the other hand, don't have the same effect as statements with free variables. Let's say we have two variables to consider, x_1 and x_2, and the world is as before.

The flow of information iii

$$[\![\exists x_1[\text{run}(x_1)]]\!]^M = \left\{ \begin{array}{l} j\ j, j\ b, b\ j, \\ b\ b, b\ a, a\ b, \\ a\ a, j\ a, a\ j \end{array} \right\}$$

$$[\![\exists x_2[\text{swim}(x_2)]]\!]^M = \left\{ \begin{array}{l} j\ j, j\ b, b\ j, \\ b\ b, b\ a, a\ b, \\ a\ a, j\ a, a\ j \end{array} \right\}$$

The flow of information iv

- Existential statements then, take all possible assignments, and just return them if the existential statement is true in the model, and reject them otherwise.

$$[\![\exists x_1[\text{run}(x_1)] \wedge \exists x_2[\text{swim}(x_2)]]\!]^M = \left\{ \begin{array}{l} j\ j, j\ b, b\ j, \\ b\ b, b\ a, a\ b, \\ a\ a, j\ a, a\ j \end{array} \right\}$$

- If nobody runs, there is no assigment g which will make the conjunctive statement true, so:

$$[\![\exists x_1[\text{run}(x_1)] \wedge \exists x_2[\text{swim}(x_2)]]\!]^M = \varnothing$$

The flow of information v

- Generally speaking, thinking of formula of FOL as sets of assignments helps us understand the following intuitive generalizations:
- Formulae with free variables typically *reduce uncertainty* about intended reference.
- Formulae with no free variables don't reduce uncertainty about intended reference.
- Intuitively, this links up with the *impossibility*, in a static setting, of an existential quantifier binding a variable outside of it's scope.

Back to DPL

- In FOL, the interpretation of a formula in a given model can be thought of as a set of assignments g – those which make the formula true.
- In DPL, the interpretation of a formula in a given model is rather going to be sets of *pairs* of assignments $\langle i, o \rangle$.
- i represents an input assignment, and o represents the output assignment resulting from the interpretation procedure.

Dynamic Existential Quantification

$$[\![\exists x\, \phi]\!] = \{\, \langle i, o\rangle \mid \exists k : k[x]i \text{ and } \langle k, o\rangle \in [\![\phi]\!] \,\}$$

- Existential quantification then, takes an input assignment i and returns an output assignment o, where o is the result of interpreting ϕ relative to a shifted assignment k.
- Don't worry too much about exactly how this works. We'll go through some derivations in detail in a little while.
- Since assignments represent the *context of utterance*, existentials are interpreted as *instructions for updating the context of utterance*.

Sets vs. functions

- Recall, we've been representing *relations*, like **hugs**, as *sets of ordered pairs*, i.e. $\{ \langle x, y \rangle \mid x \text{ hugs } y \}$.
- Intuitively then, dynamic sentence meanings are *relations* between assignments.
- Another way of expressing a relation, is as a function from two arguments to a truth value, i.e.

$$\lambda x \,.\, \lambda y \,.\, \begin{cases} 1 \text{ if } x \text{ hugs } y \\ 0 \text{ otherwise} \end{cases}$$

- This expresses the *same information* as the set of ordered pairs.

Equivalence

- We can translate then, from formulae of DPL meanings to Gennaro's dynamic calculus, and back again.

(58) Someone^{x_1} left.

$$\{ \langle i, o \rangle \mid \exists k \,:\, k[x_1]i \text{ and } k = o \text{ and } \mathrm{left}(o(x_1)) \}$$

$$\equiv$$

$$\lambda\omega \,.\, \lambda\omega' \,.\, \exists k[k[x_1]\omega \wedge k = \omega' \wedge \mathrm{left}(\omega'(x_1))]$$

- These are just two different ways of expressing the same abstract concept – a relation between assignments.

- If π is a unary predicate and α is a term, then
  $[\![\pi(\alpha)]\!] = \{\, \langle i, o\rangle \mid i = o \text{ and } [\![\alpha]\!]^o \in [\![\pi]\!] \,\}$.
- If π is a binary predicate and α and β terms, then
  $[\![\pi(\alpha, \beta)]\!] = \{\, \langle i, o\rangle \mid i = o \text{ and } \langle [\![\alpha]\!]^o, [\![\beta]\!]^o\rangle \in [\![\pi]\!] \,\}$.

Exercise

- Assume a model where:
  - $D = \{ \text{Jeff}, \text{Annie}, \text{Britta} \}$
  - Jeff hugs everyone, everyone hugs themselves, and nobody hugs anyone else.
  - Only Jeff is happy.
  - There are two variables x_1 and x_2.
- Compute the meanings of the following formulae as sets of ordered pairs of assignments:

(59) $\text{hugs}(x_1, x_2)$

(60) $\text{happy}(\text{Annie})$

(61) $\text{hugs}(\text{Jeff}, x_2)$

Example solution

$$[\![\text{hugs}(x_1, x_1)]\!]^M =$$
$$\{\, \langle i, o\rangle \mid i = o \text{ and } \langle o(x_1), o(x_2)\rangle \in I(\text{hugs}) \,\}$$

- If we're in a model with Jeff, Britta, and Annie, where everybody hugs themselves, and nobody hugs anybody else, this is the following set:

$$\left\{ \begin{array}{c} \langle [j\ j], [j\ j]\rangle, \\ \langle [b\ b], [b\ b]\rangle, \\ \langle [a\ a], [a\ a]\rangle \end{array} \right\}$$

"[..] atomic formulas do not have dynamic effects of their own. Rather, they function as a kind of"test" on incoming assignments. An atomic formula tests whether an input assignment satisfies the condition it embodies. If so, the assignment is passed on as an output, if not, it is rejected." (G&S, 1991)

Exercise

$$[\![\exists x\, \phi]\!] = \{\, \langle i, o\rangle \mid \exists k : k[x]i \text{ and } \langle k, o\rangle \in [\![\phi]\!] \,\}$$

- Compute the meaning of the following formula:

(62) $\exists x[\text{hugs}(x_1, \text{Annie})]$

- If π is a binary predicate and α and β terms, then $[\![\pi(\alpha, \beta)]\!] = \{\, \langle i, o\rangle \mid i = o \text{ and } \langle [\![\alpha]\!]^o, [\![\beta]\!]^o\rangle \in [\![\pi]\!] \,\}$.

Solution

- The set of possible assignments:

$$\left\{ \begin{array}{l} j\ j, j\ b, b\ j \\ b\ b, b\ a, a\ b \\ a\ a, a\ j, j\ a \end{array} \right\}$$

Solution ii

$$
\begin{aligned}
&[\![\exists x_1[\text{hugs}(x_1, \text{Annie})]]\!] \\
&= \{\, \langle i, o\rangle \mid \exists k[x_1]i \text{ and } \langle k, o\rangle \in [\![\text{hugs}(x_1, \text{Annie})]\!] \,\} \\
&= \{\, \langle i, o\rangle \mid \exists k[x_1]i \text{ and } \langle k, o\rangle \in \{\, \langle i', o'\rangle \mid i' = o' \text{ and } \langle o'(x_1), \text{Annie}\rangle \in \text{hugs} \,\}\,\} \\
&= \left\{ \langle i, o\rangle \;\middle|\; \exists k[x_1]i \text{ and } \langle k, o\rangle \in \left\{ \begin{array}{c} \langle [j\ j], [j\ j]\rangle, \langle [j\ b], [j\ b]\rangle, \\ \langle [a\ b], [a\ b]\rangle, \langle [a\ a], [a\ a]\rangle, \\ \langle [a\ j], [a\ j]\rangle, \langle [j\ a], [j\ a]\rangle \end{array} \right\} \right\} \\
&= \left\{ \begin{array}{c} \langle [j\ j], [j\ j]\rangle, \langle [j\ j], [a\ j]\rangle, \langle [j\ b], [j\ b]\rangle, \langle [j\ b], [a\ b]\rangle \\ \langle [b\ j], [j\ j]\rangle, \langle [b\ j], [a\ j]\rangle, \langle [b\ b], [j\ b]\rangle, \langle [b\ b], [a\ b]\rangle \\ \langle [b\ a], [j\ a]\rangle, \langle [b\ a], [a\ a]\rangle, \langle [a\ b], [a\ b]\rangle, \langle [a\ b], [j\ b]\rangle \\ \langle [a\ a], [a\ a]\rangle, \langle [a\ a], [j\ a]\rangle, \langle [a\ j], [a\ j]\rangle, \langle [a\ j], [j\ j]\rangle \\ \langle [j\ a], [j\ a]\rangle, \langle [j\ a], [a\ a]\rangle \end{array} \right\}
\end{aligned}
$$

Solution iii: the result reformatted

$$[\![\exists x_1[\text{hugs}(x_1, \text{Annie})]]\!] = \left\{ \begin{array}{l} \langle [j\ j], [j\ j] \rangle \\ \langle [b\ j], [j\ j] \rangle \\ \langle [a\ j], [j\ j] \rangle \\ \langle [j\ j], [a\ j] \rangle \\ \langle [b\ j], [a\ j] \rangle \\ \langle [a\ j], [a\ j] \rangle \\ \langle [j\ a], [j\ a] \rangle \\ \langle [b\ a], [j\ a] \rangle \\ \langle [a\ a], [j\ a] \rangle \\ \langle [j\ b], [j\ b] \rangle \\ \langle [b\ b], [j\ b] \rangle \\ \langle [a\ b], [j\ b] \rangle \\ \langle [j\ b], [a\ b] \rangle \\ \langle [b\ b], [a\ b] \rangle \\ \langle [a\ b], [a\ b] \rangle \\ \langle [j\ a], [a\ a] \rangle \\ \langle [b\ a], [a\ a] \rangle \\ \langle [a\ a], [a\ a] \rangle \end{array} \right\}$$

Solution iv

- Existential quantification is *dynamic*, since it potentially changes the input assignment.
- We can see more clearly what the existential statement does by gathering together the set of input assignments and the set of output assignments:

$$\left\langle \left\{ \begin{array}{l} j\ j, j\ b, b\ j \\ b\ b, b\ a, a\ b \\ a\ a, a\ j, j\ a \end{array} \right\}, \left\{ \begin{array}{l} j\ j, j\ b, j\ a \\ a\ a, a\ b, a\ j \end{array} \right\} \right\rangle$$

- $\exists x_1[\mathrm{hugs}(x_1, \mathrm{Annie})]$ takes all assignments as its input, and outputs those assignments where x_1 is mapped to someone who hugged Annie – namely, Jeff or Annie.

Dynamic conjunction

- Unlike in classical semantics, we now have a theory of existential quantification according to which it *reduces uncertainty* about intended reference.
- But hang on, we still need to say something about conjunction DPL to derive cross-sentential anaphora.
- Here is the rule for conjunction in DPL:
- $[\![[\phi \wedge \psi]]\!] = \{ \langle i, o \rangle \mid \exists k : \langle i, k \rangle \in [\![\phi]\!] \text{ and } \langle k, o \rangle \in [\![\psi]\!] \}$
- Our definition for dynamic conjunction evaluates the first conjunct ϕ relative to the input assignment i, resulting in k, then threads k into ψ.

Dynamic conjunction ii

- We can assume that successive sentences are interpreted as *conjoined*. This will give us a uniform account of anaphora across conjoined clauses, and cross-sentential anaphora.

(63) Someone$^{x^1}$ hugged Annie. They$_{x_1}$ are happy.

(64) $\exists x_1[\textbf{hugged}(x_1, \textbf{Annie})] \wedge \textbf{happy}(x_1)$

- Recall that we're in a model where everyone only hugs themselves, except for Jeff, who also hugs Annie, and only Jeff is happy.

Dynamic conjunction iii

- We know how the first conjunct is evaluated:

$$[\![\exists x_1[\text{hugged}(x_1, \text{Annie})]]\!]$$

$$= \left\{ \begin{array}{r} \langle [j\ j], [j\ j] \rangle, \langle [j\ j], [a\ j] \rangle, \langle [j\ b], [j\ b] \rangle, \langle [j\ b], [a\ b] \rangle \\ \langle [b\ j], [j\ j] \rangle, \langle [b\ j], [a\ j] \rangle, \langle [b\ b], [j\ b] \rangle, \langle [b\ b], [a\ b] \rangle \\ \langle [b\ a], [j\ a] \rangle, \langle [b\ a], [a\ a] \rangle, \langle [a\ b], [a\ b] \rangle, \langle [a\ b], [j\ b] \rangle \\ \langle [a\ a], [a\ a] \rangle, \langle [a\ a], [j\ a] \rangle, \langle [a\ j], [a\ j] \rangle, \langle [a\ j], [j\ j] \rangle \\ \langle [j\ a], [j\ a] \rangle, \langle [j\ a], [a\ a] \rangle \end{array} \right\}$$

Dynamic conjunction iv

- And we know how the second conjunct should be evaluated:

$$\begin{aligned}&[\![\text{happy}(x_1)]\!]\\ &= \left\{ \langle [j\ j], [j\ j] \rangle, \langle [j\ b], [j\ b] \rangle, \langle [j\ a], [j\ a] \rangle \right\}\end{aligned}$$

- If π is a unary predicate and α is a term, then $[\![\pi(\alpha)]\!] = \{ \langle i, o \rangle \mid i = o \text{ and } [\![\alpha]\!]^o \in [\![\pi]\!] \}$.

Dynamic conjunction v

- $[\![[\phi \wedge \psi]]\!] = \{ \langle i, o \rangle \mid \exists k : \langle i, k \rangle \in [\![\phi]\!] \text{ and } \langle k, o \rangle \in [\![\psi]\!] \}$

$$[\![\exists x_1[\text{hugged}(x_1, \text{Annie})] \wedge \text{happy}(x_1)]\!]$$

$$= \left\{ \begin{array}{r} \langle [j\ j], [j\ j] \rangle, \langle [j\ b], [j\ b] \rangle \\ \langle [b\ j], [j\ j] \rangle, \langle [b\ b], [j\ b] \rangle \\ \langle [b\ a], [j\ a] \rangle, \langle [a\ b], [j\ b] \rangle \\ \langle [a\ a], [j\ a] \rangle, \langle [a\ j], [j\ a] \rangle \\ \langle [j\ a], [j\ a] \rangle \end{array} \right\}$$

- The prediction is, that *someone hugged Annie. They are happy.* should update a context where we're uncertain about who x_1 picks out, to one where we're certain about who x_1 picks out – namely, Jeff.

Associativity of $\exists v$ and $\wedge$

- In fact, $\exists x_1[\text{hugged}(x_1, \text{Annie})] \wedge \text{happy}(x_1)$ turns out to be *equivalent* to the formula: $\exists x_1[\text{hugged}(x_1, \text{Annie}) \wedge \text{happy}(x_1)]$
- *Exercise:* Compute the meaning of each of the conjuncts in DPL, and compute the result of dynamic conjunction:
- $[\![[\phi \wedge \psi]]\!] = \{ \langle i, o \rangle \mid \exists k : \langle i, k \rangle \in [\![\phi]\!] \text{ and } \langle k, o \rangle \in [\![\psi]\!] \}$

Binding failure

- Let's double check that this is genuinely binding. What happens when the existential and the pronoun in the second clause are contra-indexed?

(65) $\exists x_1[\text{hugged}(x_1, \text{Annie})] \wedge \text{happy}(x_2)$

Binding failure ii

- Again, we know how the first conjunct should be evaluated:

$$[\![\exists x_1[\text{hugged}(x_1, \text{Annie})]]\!]$$

$$= \left\{ \begin{array}{r} \langle [j\ j], [j\ j] \rangle, \langle [j\ j], [a\ j] \rangle, \langle [j\ b], [j\ b] \rangle, \langle [j\ b], [a\ b] \rangle \\ \langle [b\ j], [j\ j] \rangle, \langle [b\ j], [a\ j] \rangle, \langle [b\ b], [j\ b] \rangle, \langle [b\ b], [a\ b] \rangle \\ \langle [b\ a], [j\ a] \rangle, \langle [b\ a], [a\ a] \rangle, \langle [a\ b], [a\ b] \rangle, \langle [a\ b], [j\ b] \rangle \\ \langle [a\ a], [a\ a] \rangle, \langle [a\ a], [j\ a] \rangle, \langle [a\ j], [a\ j] \rangle, \langle [a\ j], [j\ j] \rangle \\ \langle [j\ a], [j\ a] \rangle, \langle [j\ a], [a\ a] \rangle \end{array} \right\}$$

Binding failure iii

- The second conjunct is evaluated differently – it is a *test* on a context where x_2 picks out someone who is happy.

$$\begin{aligned}&[\![\text{happy}(x_2)]\!]\\&= \left\{ \langle [j\ j], [j\ j] \rangle, \langle [a\ j], [a\ j] \rangle, \langle [b\ j], [b\ j] \rangle \right\}\end{aligned}$$

- If π is a unary predicate and α is a term, then $[\![\pi(\alpha)]\!] = \{ \langle i, o \rangle \mid i = o \text{ and } [\![\alpha]\!]^o \in [\![\pi]\!] \}$.

Binding failure iv

- $[\![[\phi \wedge \psi]]\!] = \{ \langle i, o \rangle \mid \exists k : \langle i, k \rangle \in [\![\phi]\!] \text{ and } \langle k, o \rangle \in [\![\psi]\!] \}$

$$[\![\exists x_1[\text{hugged}(x_1, \text{Annie})] \wedge \text{happy}(x_2)]\!]$$

$$= \left\{ \begin{array}{l} \langle [j\ j], [j\ j] \rangle \\ \langle [j\ j], [a\ j] \rangle \\ \langle [b\ j], [j\ j] \rangle \\ \langle [b\ j], [a\ j] \rangle \\ \langle [a\ j], [a\ j] \rangle \\ \langle [a\ j], [j\ j] \rangle \end{array} \right\}$$

- Great! this sentence is correctly interpreted to *reduce certainty* about who x_1 picks out *and* to act as a test on who x_2 picks out.

Cross-sentential crossover i

- Remember, one of the candidate Logical Forms for cross-sentential anaphora we considered involved a genuinely wide-scope existential.

(66) Someone$^{\mathbf{1}}$ hugged Annie and they$_{\mathbf{1}}$ are happy.

(67) $\exists x_1[\text{hugged}(x_1, \text{Annie}) \wedge \text{happy}(x_1)]$

- If we were to allow such a Logical Form, we need to rule out the following:

(68) They$_{\mathbf{1}}$ are happy and someone^{1} hugged Annie.

- Why can't this receive the following Logical Form, and get a sensible interpretation?

(69) $\exists x_1[\text{happy}(x_1) \wedge \text{hugged}(x_1, \text{Annie})]$

Cross-sentential crossover ii

- DPL (and dynamic semantics more generally) solves this completely straightforwardly.
- According to Dynamic Semantics, the Logical Form should really be the following:

(70) $\mathbf{happy}(x_1) \wedge \exists x_1[\mathbf{hugged}(x_1, \mathbf{Annie})]$

- Because of how dynamic conjunction works, binding won't obtain.

Cross-sentential crossover iii

- Again, we know how the first conjunct is interpreted:

$$\begin{aligned}&[\![\text{happy}(x_1)]\!]\\&= \left\{ \langle [j\ j], [j\ j] \rangle, \langle [j\ b], [j\ b] \rangle, \langle [j\ a], [j\ a] \rangle \right\}\end{aligned}$$

Cross-sentential crossover iv

- We know how the second conjunct is interpreted:

$$[\![\exists x_1[\text{hugged}(x_1, \text{Annie})]]\!]$$

$$= \left\{ \begin{array}{r} \langle [j\ j], [j\ j] \rangle, \langle [j\ j], [a\ j] \rangle, \langle [j\ b], [j\ b] \rangle, \langle [j\ b], [a\ b] \rangle \\ \langle [b\ j], [j\ j] \rangle, \langle [b\ j], [a\ j] \rangle, \langle [b\ b], [j\ b] \rangle, \langle [b\ b], [a\ b] \rangle \\ \langle [b\ a], [j\ a] \rangle, \langle [b\ a], [a\ a] \rangle, \langle [a\ b], [a\ b] \rangle, \langle [a\ b], [j\ b] \rangle \\ \langle [a\ a], [a\ a] \rangle, \langle [a\ a], [j\ a] \rangle, \langle [a\ j], [a\ j] \rangle, \langle [a\ j], [j\ j] \rangle \\ \langle [j\ a], [j\ a] \rangle, \langle [j\ a], [a\ a] \rangle \end{array} \right\}$$

Cross-sentential crossover v

- $[\![[\phi \wedge \psi]]\!] = \{ \langle i, o \rangle \mid \exists k : \langle i, k \rangle \in [\![\phi]\!] \text{ and } \langle k, o \rangle \in [\![\psi]\!] \}$
- We're only allowed to take *input-output* pairs, where the output of threading the input into ϕ creates a valid input for ψ.

$$[\![\text{happy}(x_1) \wedge \exists x_1[\text{hugged}(x_1, \text{Annie})]]\!]$$

$$= \left\{ \begin{array}{l} \langle [j\ j], [j\ j] \rangle, \langle [j\ j], [a\ j] \rangle, \\ \langle [j\ b], [j\ b] \rangle, \langle [j\ b], [a\ b] \rangle \\ \langle [j\ a], [j\ a] \rangle, \langle [j\ a], [a\ a] \rangle \end{array} \right\}$$

Cross-sentential crossover vi

- if x_1 were bound, then the effect of the sentence on the context should be to fully reduce uncertainty about who x_1 is intended to pick out – namely Jeff.
- Instead, the sentence is interpreted as a transition from a context in which we *know* that x_1 is intended to pick out Jeff, to one where x_1 is "re-opened", and could pick out anyone who hugs Annie.
- This is because the meaning given for dynamic conjunction is *inherently* asymmetric and specifically left-to-right: first we thread the input into the first conjunct, and then we thread the result into the second conjunct.
- $[\![[\phi \wedge \psi]]\!] = \{ \langle i, o \rangle \mid \exists k : \langle i, k \rangle \in [\![\phi]\!] \text{ and } \langle k, o \rangle \in [\![\psi]\!] \}$

Logical properties of DPL

- So, we've shown that, in DPL, conjunction is asymmetric:
- $[\![\phi \wedge \psi]\!] \not\equiv [\![\psi \wedge \phi]\!]$
- It's worth stressing just how much of a radical departure from classical semantics this is!
- In DPL, the dynamic, left-to-right nature of conjunction is really built into the *semantics* of the operator – this is by no means a byproduct of a pragmatic mechanism.

External vs. internal dynamicity

- Conjunction passes on variable binding from its left conjunct to its right.
- In DPL, connectives of this kind are called *internally dynamic*. This is out basic case: *Someone walked in and they sat down*.
- A conjunctive formula can keep passing on variable bindings to conjuncts yet to come, i.e. *Someone walked in and they sat down. They got a drink.*
- In DPL, connectives of this kind are called *externally dynamic*.

Donkey sentences

- We still haven't given a treatment of donkey sentences.

(71) If a farmer owns a donkey^{y}, he$_{x}$ hits it$_{y}$.

- Remember that, in FOL, the correct translation would seem to involve universal quantification.

$$\forall x, y[\text{farmer}(x) \wedge \text{donkey}(y) \wedge \text{own}(x, y) \rightarrow \text{hits}(x, y)]$$

- Unlike previously, the trick of scoping out an existential quantifier doesn't have a chance of working, since we'd end up talking about a specific farmer and donkey.

Donkey sentences ii

(72) *If a farmer owns a donkey^{y}, he$_{x}$ hits it$_{y}$. It$_{y}$ yelps.

(73) *If a farmer owns a donkey^{y}, he$_{x}$ hits it$_{y}$. he$_{y}$ gets thirsty.

- Is *implication* ($\rightarrow$)..
  - externally dynamic?
  - internally dynamic?

Exercise

- For *disjunction* ($\vee$), come up with data which show whether it is...
  - externally dynamic?
  - internally dynamic?

Dynamic implication

- The fact that implication is not externally dynamic, means that overall, an implciational formula should be a *test*, i.e., it shouldn't alter the input context.
- But, it should pass variable binding on from the antecedent to the consequent.
- Here is the entry for dynamic implication:
- $[\![\phi \rightarrow \psi]\!] =$
  $\{ \langle i, o \rangle \mid o = i \text{ and } \forall k : \langle o, k \rangle \in [\![\phi]\!] \Rightarrow \exists j : \langle k, j \rangle \in [\![\psi]\!] \}$
- An implicational statement is a *test* on an assignment o, where every result of threading o into the antecedent can be threaded into the consequent.

Donkey derivation

(74) If a farmer^1 owns a donkey^2, he$_1$ hits it$_2$.

(75) $(\exists x_1, x_2[\text{farmer}(x_1) \wedge \text{donkey}(x_2) \wedge \text{owns}(x_1, x_2)]) \rightarrow \text{hits}(x_1)(x_2)$

- $\texttt{farmer} = \{ a, b \}$
- $\texttt{donkey} = \{ c, d, e \}$
- a owns c, b owns d and e.

Donkey derivation ii

- First let's compute the meaning of the antecedent.

$$[\![\exists x_1, x_2[\text{farmer}(x_1) \wedge \text{donkey}(x_2) \wedge \text{owns}(x_1, x_2)]]\!]$$

- Informally, this will end up expressing a transition from an *input* to an output that maps x_1 to a farmer, and x_2 to a donkey that he owns.

$$\{ \langle [a\ c], [a\ c] \rangle, \langle [a\ d], [a\ c] \rangle, \langle [a\ e], [a\ c] \rangle$$
$$\langle [b\ c], [a\ c] \rangle, \langle [b\ b], [b\ e] \rangle$$
$$\ldots$$

etc.

Donkey derivation iii

- Holding fixed that x_1 is a farmer and x_2 is a donkey:

$$\left\langle \left\{ \begin{array}{r} [a\ c], [a\ d],\ [a\ e], \\ [b\ c],\ [b\ d],\ [b\ e], \\ \ldots \end{array} \right\}, \left\{ \begin{array}{c} [a\ c], \\ [b\ d], \\ [b\ e] \end{array} \right\} \right\rangle$$

Donkey derivation iv

- The meaning of the consequent is just a test on assignments where x_1 owns x_2:

$$\{ \langle [a\ c], [a\ c] \rangle, \langle [b\ d], [b\ d] \rangle, \langle [b\ e], [b\ e] \rangle \}$$

- $[\![\phi \to \psi]\!] =$
  $\{ \langle i, o \rangle \mid o = i \text{ and } \forall k : \langle o, k \rangle \in [\![\phi]\!] \Rightarrow \exists j : \langle k, j \rangle \in [\![\psi]\!] \}$
- So the implicational statement as a whole is a *test*, where we inspect the outputs of the antecedent, and ensure that they're valid inputs to the consequent.

Weak vs. strong donkeys

- DPL generates a so-called *strong* reading for donkey anaphora.

(76) If a farmer owns a donkey, he beats it.

- In DPL, this expresses that a farmer beats *every* donkey that he owns – in the previous example, b must beat d and e for the test to pass.
- However, there are contexts where it looks like we need to be able to derive a *weak* reading.

Weak donkeys

(77) Yesterday, every person who had a credit card^{y} paid his bill with it$_{y}$.
(R. Cooper, according to Chierchia 1995)

- The most salient reading of the above does not require each person to use every credit card they have to pay their bill, rather each person must use at least one of their credit cards to pay their bill.
- Traditional dynamic semantic accounts, such as DPL, don't derive this *weak* donkey reading.

Negation

- Consider the following.

(78) *It's not the case that someone^{x} hugged Annie. They$_{x}$ sat down.

(79) *Nobody^{x} hugged Annie. They$_{x}$ sat down.

- Negation *blocks* dynamic binding, i.e., it closes off the anaphoric potential introduced by the existential.

Negation ii

- Here is our rule for dynamic negation:
- $[\![\neg\phi]\!] = \{ \langle i, o\rangle \mid o = i \wedge \neg\exists k : \langle o, k\rangle \in [\![\phi]\!] \}$
- Negation is a *test* on assignments o, such that feeding o into ϕ gives rise to no outputs.
- Since negation is a test, it must be externally static.
- Let's check that negation blocks binding.

Negation iii

(80) It's not the case that someone hugged Annie. They$_{x}$ sat down.

- If at least one person hugged Annie, the first sentence will always return $\varnothing$.
- If nobody hugged Annie, the first sentence will just be a test on the set of assignments.
- *Exercise:* show this by going through the computation.

Problems for dynamic semantics

(81) If a client turns up, you treat him$_{x}$ politely. You offer him$_{x}$ a cup of coffee and his him$_{x}$ to wait.

(82) Every player^{y} chooses a pawn. He$_{y}$ puts it$_{x}$ on square one.

(83) It is not true that John doesn't own a car. It$_{x}$ is red, and it$_{x}$ is parked in from of his house.

(84) Either there is no bathroom here, or it is in a funny place. In any case, it is not on the first floor.

- All examples from G&S 1991.

Lecture 3: Discourse referents

Outline

- In the first part of the class, we'll explore the parallel between dynamic sentence meanings, and the meanings of computer programs in more detail.
- In the second part of the class, we'll formulate an alternative dynamic semantics, based on Dekker's *Predicate Logic with Anaphora*, according to which indefinites add discourse referents to a *stack*.
- This will result in an arguably more elegant system in which the notion of *discourse referent* is reified.

Variables in programming languages

- In the vast majority of programming languages, *variables*, such as `x`, `y`, `z`, etc., can be used as placeholders for values.
- Consider the following program `main` (written in the `Rust` programming language).
- Ignoring the boilerplate, `main` introduces a variable named `x`, sets `x`'s value to the integer `5`, and prints the result of `x + 1`.

```
fn main() {
    let x = 5;
    println!("{}", x + 1);
    }
```

- Q: what value gets printed when we run `main`? Let's find out...

Variables ii

- Unsurprisingly, `main` returns `6` – in the argument to `println!`, `x` is replaced by the stored value `5`:

```
println!("{}", x + 1)
println!("{}", 5 + 1)
println!("{}", 6)
```

Variables iii

- Now let's change `main` a bit.

```
fn main() {
    let x = 5;
    x = x + 4;
    println!("{}", x + 1);
    }
```

- what happens when we run `main`?

Variables iv

- Whoops! the `Rust` compiler throws out an error:

```
let x = 5;
    - first assignment to `x`

x = x + 4;
^^^^^^^^^ cannot assign twice to immutable variable
```

- what went wrong?

(Im)mutability

- In `Rust` (and several other programming languages), variables are by default `immutable` (i.e., unchangeable). This means that they are simply *names for values*.
- It simply doesn't make sense to write `x = x + 4`, since we're essentially assigning two different values to `x`.
- What we need in order to make sense of this program is the concept of a *mutable variable* – rather than acting as a name for a value, we want our variable to act as an *address for a potentially changeable value*.
- Conveniently for our purposes, variables in `Rust` can be rendered mutable via the `mut` keyword.
- OK, let's try that again:

(Im)mutability II

- Let's try running `main`, but now let's explicitly state that `x` is *mutable*.

```
fn main() {
  let mut x = 5;
  x = x + 4;
  println!("{}", x + 1);
  }
```

- returns: `10`!

- Let's see what's going on here in a little more detail.

(Im)mutability III

First we introduce a *mutable* variable `x`, and set its value to `5`.

```
let mut x = 5;
```

Next we retrieve `x`, and set its new value to its old value (`5`) `+ 1`.

```
x = x + 4
x = 5 + 4
x = 9
```

Now we retrieve `x` and print the result of `x + 1`

```
println!("{}", x + 1)
println!("{}", 9 + 1)
println!("{}", 10)
println!("10")
```

Order sensitivity

Once we introduce mutability, our programs become *order sensitive*:

```
let mut x = 5;
x = x * 2
x = x - 1
println!("{}", x)
```

prints: `9`

```
let mut x = 5;
x = x - 1
x = x * 2
println!("{}", x)
```

prints: `8`

Back to natural language

- Think of a number.
- Now, multiply it by two.
- Now, subtract one from it.
- Now, tell me it.

Back to natural language

- Think of a number

  `let mut x = 5`.

- Now, multiply it by two.

  ```
  x = x * 2
  x = 10
  ```

- Now, subtract one from it.

  ```
  x = x - 1
  x = 9
  ```

- Now, tell me it.

  `println!("9")`

Back to natural language

- Whoa - we've just made a major discovery! Natural language makes use of (something like) *mutable variables*, otherwise the preceding discourse wouldn't make sense.
- We can think of indefinites like *a number* as introducing new *mutable variables*.
- Pronouns, like *it*, refer back to an already-introduced variable.
- The value assigned to a variable can change over the course of a discourse.

Karttunen's discourse referents

- The idea that indefinites introduce mutable variables is an insight originally due to semanticist and computational linguist Lauri Karttunen (although he doesn't use this idiom).

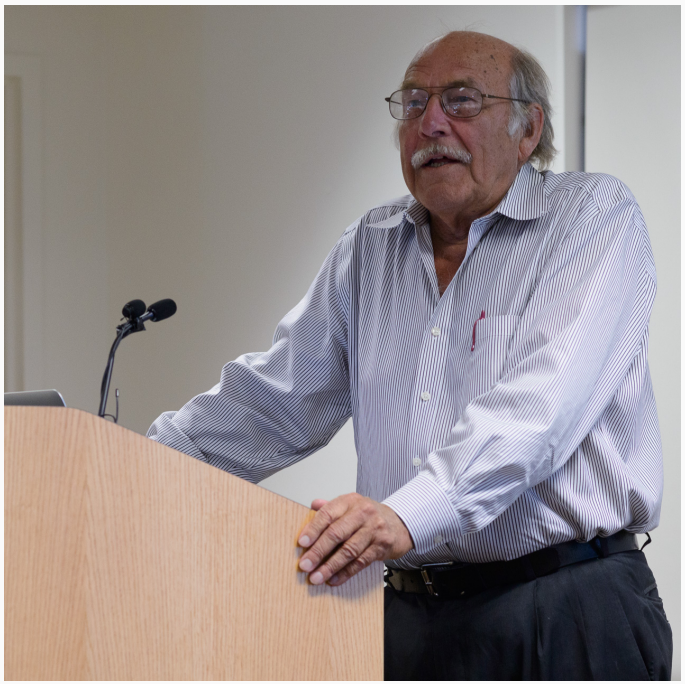


Figure 9: Lauri Karttunen (src: Stefan Müller)

Karttunen's discourse referents ii

"Consider a device designed to read a text in some natural language, interpret it, and store the content in some manner, say, for the purpose of being able to answer questions about it. To accomplish this task, the machine will have to fulfill at least the following basic requirement. It has to be able to build a file that consists of records of all the individuals, that is, events, objects, etc., mentioned in the text and, for each individual, record whatever is said about it." (Karttunen 1976)

Cont.

“I intend to discuss one particular feature a text interpreter must have: that it must be able to recognize when a novel individual is mentioned in the input text and to store it along with its characterization for future reference.”

Cont

"We found that in simple sentences [...] an indefinite NP establishes a discourse referent just in case the sentence is an affirmative assertion. By"establishes a discourse referent", we meant that there may be a coreferential pronoun or definite noun phrase later in the discourse."

DyS

Overview

- In our previous logical system, DPL, formulae could either:
    - Act as tests on input assignments, returning the same assignments.
    - Shrink the set of assignments, taking a set of assignments, and returning a subset.
- The latter case represented the contribution of existentially quantified sentences.
- The notion of a *discourse referent* has no direct correlate.

PLA via DyS

- PLA simplifies things somewhat by having semantic objects which correspond directly to *discourse referents*.
- N.b. the version of PLA I'm presenting is simplified relative Dekker (1994), and corresponds more closely to Charlow (2014)'s DyS.
- The basic ideas are the same, but unlike PLA, DyS doesn't make use of *assignments*. This makes it a little easier to reason about.

Stacks

- The current state of the discourse is represented as a *stack*, which is just going to be a (potentially empty) sequence of objects.

$$s = a...xyz$$

- The objects in the stack represent the *discourse referents* which have been introduced over the course of the discourse so far.

Stacks ii

- Stacks can be *extended* with additional drefs, simply by adding them to (the end of) the stack.

$$s = jb$$

- We define a primitive binary operation that takes a stack s, and an object m, and pushes m to s.

$$\widehat{sm} = hbm$$

Stacks iii

- We're also going to define a primitive unary operation τ, which returns the last element to be added to a stack.

$$s' = hbm$$

$$s'_\tau = m$$

Syntax of DyS

- For simplicity, I'll take the syntax of DyS to be identical to the syntax of FOL (and hence DPL).
- With one exception - we have a new syntactic expression *pro*, which has the same distribution as individual constants and variables – namely it can appear as an argument to predicates, e.g.,
- $\text{hugs}(\text{Annie}, \textit{pro})$
- Unlike in DPL, sentence meanings in DyS are going to be additionally relativised to an assignment g. We'll see the relevance of this later.

Semantics of DyS

- Sentence meanings in DyS are going to express *relations between stacks*, rather than relations between assignment functions.
- Here's the interpretation rule for a unary predicate:
- If π is a unary predicate and α is an individual constant, then $[\![\pi(\alpha)]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{s[\![\alpha]\!]^g} \text{ and } [\![\alpha]\!]^g \in [\![\pi]\!] \right\}$
- If π is a unary predicate and α is a variable, then $[\![\pi(\alpha)]\!]^g = \left\{ \langle s, s' \rangle \mid s' = s \text{ and } [\![\alpha]\!]^g \in [\![\pi]\!] \right\}$

- For binary predicates we now technically need four rules, since individual constants update the stack, whereas variables don't quantifiers don't:

- If π is a binary predicate and α, β are individual constants, then
$$[\![\pi(\alpha,\beta)]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{s[\![\beta]\!]^g[\![\alpha]\!]^g} \text{ and } \langle [\![\alpha]\!]^g, [\![\beta]\!]^g \rangle \in [\![\pi]\!]^g \right\}$$

- If π is a binary predicate and α is an individual constants, and β is a variable, then
$$[\![\pi(\alpha,\beta)]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{s[\![\alpha]\!]^g} \text{ and } \langle [\![\alpha]\!]^g, [\![\beta]\!]^g \rangle \in [\![\pi]\!]^g \right\}$$

- If π is a binary predicate and α is a variable, and β is an individual constant, then
$$[\![\pi(\alpha,\beta)]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{s[\![\beta]\!]^g} \text{ and } \langle [\![\alpha]\!]^g, [\![\beta]\!]^g \rangle \in [\![\pi]\!]^g \right\}$$

- If π is a binary predicate and α and β are variables, then
$$[\![\pi(\alpha,\beta)]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; s' = s \text{ and } \langle [\![\alpha]\!]^g, [\![\beta]\!]^g \rangle \in [\![\pi]\!]^g \right\}$$

Semantics of Dys

- The general rule is: *individual constants* trigger a push to the output stack, whereas *variables* do not.

Semantics of DyS

- Assume a model with *Jeff*, *Britta*, and *Annie*.
- Only *Jeff is happy*, and nobody else is happy.
- *Everyone hugged themselves*, and nobody else hugged anybody else.
- Task: compute the interpretation of the following formulas:

(85) $\mathrm{happy}(\mathrm{Jeff})$

(86) $\mathrm{hugs}(\mathrm{Annie}, \mathrm{Annie})$

$$
\begin{aligned}
&[\![\text{happy(Jeff)}]\!]^g \\
&= \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{sj} \wedge j \in \mathsf{happy} \right\} \\
&= \{ \langle [], [j] \rangle, \langle [x], [xj] \rangle, \langle [xy], [xyj] \rangle, \ldots \}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\text{hugs(Annie, Annie)}]\!]^g \\
&= \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{saa} \wedge \langle a, a \rangle \in \mathsf{hugs} \right\} \\
&= \{ \langle [], [aa] \rangle, \langle [x], [xaa] \rangle, \langle [xy], [xyaa] \rangle, \ldots \}
\end{aligned}
$$

Variables

- Sentences with variables are just going to be tests on stacks.

$$
\begin{aligned}
&[\![\text{happy}(x)]\!]^g \\
&= \{\, \langle s, s' \rangle \mid s' = s \wedge g(x) \in \textsf{happy} \,\} \\
&\text{if } g(x) = j \text{ then } = \{\, \langle [], [] \rangle, \langle [x], [x] \rangle, \langle [xy], [xy] \rangle, \ldots \,\} \\
&\text{else } \varnothing
\end{aligned}
$$

Falsity

- Just like in DPL, if an atomic formula is false in the model, it returns the empty set.

$$
\begin{aligned}
&\text{happy(Britta)} \\
&= \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{sb} \wedge b \in \text{happy} \right\} \\
&= \varnothing
\end{aligned}
$$

Existential quantification

- Existentially quantified statement $\exists v\, \phi$ in this setting, are interpreted as instructions to update a stack s with a random individual x, just so long as the embedded formula ϕ is true relative to g, modified so that it maps v to x.
- $[\![\exists v\, \phi]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; \exists x[s' = s\widehat{x} \wedge [\![\phi]\!]^{g^{[v \to x]}}] \right\}$
- The idea is that existential quantifiers represent a *refusal to choose between different possible referents*

Illustration

- Let's say we're in a model, again with j, b, and a, and only j and b arrived.

$$
\begin{aligned}
&[\![\exists v[\mathrm{arrive}(v)]]\!]^g \\
&= \left\{ \langle s, s' \rangle \;\middle|\; \exists x[s' = s\hat{x} \wedge [\![\mathrm{arrive}(v)]\!]^{g[x \to v]}] \right\} \\
&= \left\{ \langle s, s' \rangle \;\middle|\; \exists x[s' = s\hat{x} \wedge x \in \mathtt{arrive}] \right\} \\
&= \left\{ \begin{array}{l} \langle [], [j] \rangle \\ \langle [], [b] \rangle \\ \langle [x], [xj] \rangle \\ \langle [x], [xb] \rangle \\ \langle [xy], [xyj] \rangle \\ \langle [xy], [xyb] \rangle \end{array} \right\}
\end{aligned}
$$

Pronouns

- How do pronouns pick up referents?
- Unlike DPL, we distinguish both syntactically and semantically between pronoun binding and variable binding.
- The rule for formulas with *pro* is going to be the following:
- If π is a unary predicate, then
  $[\![\pi(pro)]\!]^g = \{ \langle s, s' \rangle \mid s' = s \text{ and } s'_\tau \in [\![\pi]\!] \}$
- The idea is that pronouns return *the last object to be added to the stack*.

Pronouns

- Of course this means we'll need to multiply our rules for binary connectives, so that we can deal with combinations of proper names and pronouns (and variables and pronouns, but I won't show that here).

- If π is a binary predicate and α is an individual constants, then $[\![\pi(\alpha, pro)]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{s[\![\alpha]\!]^g} \text{ and } \langle [\![\alpha]\!]^g, s'_\tau \rangle \in [\![\pi]\!]^g \right\}$

- If π is a binary predicate and α is an individual constant, then $[\![\pi(pro, \alpha)]\!]^g = \left\{ \langle s, s' \rangle \;\middle|\; s' = \widehat{s[\![\alpha]\!]^g} \text{ and } \langle s_\tau, [\![\alpha]\!]^g \rangle \in [\![\pi]\!]^g \right\}$

- Notice that our interpretation rules for binary predicates with *pro* are *internally dynamic* – when *pro* is the object, it picks up the referent pushed to the stack by the subject.

- Because of the way the rules are defined, the reverse doesn't go through.

Pronouns ii

(87) Annie hugged herself – $\mathbf{hugged}(\mathbf{Annie}, \mathit{pro})$

$$= \left\{ \langle s, s' \rangle \mid s' = \widehat{sa} \wedge \langle a, s'_\tau \rangle \in \mathsf{hug} \right\}$$

- This is guaranteed to be true in our model, since the pronoun picks up the discourse referent introduced by the subject.

(88) She hugged Annie. – $\mathbf{hugged}(\mathit{pro}, \mathbf{Annie})$

- The meaning here is dependent on the incoming stack s.

$$= \left\{ \langle s, s' \rangle \mid s' = \widehat{sa} \wedge \langle a, s_\tau \rangle \in \mathsf{hug} \right\}$$

Pronouns iii

- Oh look! We've derived a basic version of Condition C of the binding theory.

(89) Annie$^{\mathbf{1}}$ loves herself$_{\mathbf{1}}$.

(90) *she$_{\mathbf{1}}$ loves Annie$^{\mathbf{1}}$.

- Of course we need to refine the theory to get a broad empirical converage.

- In particular, we probably want to allow pronouns to pick up not just the last discourse referent added to the stack, to account for binding of pronouns across other NPs.

(91) Annie$^{\mathbf{1}}$ thinks that Bill$^{\mathbf{2}}$ hugged her$_{\mathbf{1}}$.

Dynamic binding

- Existential quantifiers can bind an object *pro* in exactly the same way.

$$
\begin{aligned}
&[\![\exists v[\text{hugged}(v, \mathit{pro})]]\!]^g \\
&= \left\{ \langle s, s' \rangle \;\middle|\; \exists x[s' = s\hat{x} \wedge [\![\text{hugged}(v, \mathit{pro})]\!]^{g^{[v \to x]}}] \right\} \\
&= \left\{ \langle s, s' \rangle \;\middle|\; \exists x[s' = s\hat{x} \wedge \langle x, s'_\tau \rangle \in \text{hugged}] \right\} \\
&= \left\{ \begin{array}{l} \langle [], [j] \rangle \\ \langle [], [b] \rangle \\ \langle [], [a] \rangle \\ \langle [x], [xj] \rangle \\ \langle [x], [xb] \rangle \\ \langle [x], [aa] \rangle \\ \ldots \end{array} \right\}
\end{aligned}
$$

Dynamic binding and crossover

- But not a subject *pro*!

$$
\begin{aligned}
&[\![\exists v[\text{hugged}(pro, v)]]\!]^g \\
&= \left\{ \langle s, s' \rangle \;\middle|\; \exists x[s' = s\widehat{x} \wedge [\![\text{hugged}(pro, v)]\!]^{g^{[v \to x]}}] \right\} \\
&= \left\{ \langle s, s' \rangle \;\middle|\; \exists x[s' = s\widehat{x} \wedge \langle s_\tau, x \rangle \in \text{hugged}] \right\} \\
&= \left\{ \begin{array}{l} \langle [j], [jj] \rangle \\ \langle [a], [aa] \rangle \\ \langle [b], [bb] \rangle \\ \langle [xj], [xjj] \rangle \\ \langle [xa], [xaa] \rangle \\ \langle [xb], [xbb] \rangle \\ \ldots \end{array} \right\}
\end{aligned}
$$

Dynamic relations

- Without even saying anything about conjunction, we've already captured the fact that *binary predicates are internally dynamic*, something not captured by DPL.
- We did this, essentially, by lexical stipulation, but there are ways of reformulating DyS which make this more principled.

Dynamic conjunction

- Conjunction in DyS is defined in the same way as DPL, but in terms of stacks rather than assignments:
- $[\![[\phi \wedge \psi]]\!]^g = \{ \langle s, s' \rangle \mid \exists k : \langle s, k \rangle \in [\![\phi]\!]^g \text{ and } \langle k, s' \rangle \in [\![\psi]\!] \}$
- We feed our input stack s into the first conjunct, and return the result of feeding the output into the second conjunct.

Cross-sentential binding

- Cross-sentential binding is captured in a totally straightforward way. Let's say that j hugged a, everyone hugged themselves, nobody hugged anyone else, and only j is happy.

(92) $\exists v[\text{hug}(v, \text{Annie})] \wedge \text{happy}(\textit{pro})$

$$[\![\exists v[\text{hug}(\text{v},\text{Annie})]]\!]^g$$
$$= \{ \langle s, s' \rangle \mid \exists x[s' = \widehat{sax} \wedge \langle x, a \rangle \in \text{hug}] \}$$
$$= \{ \langle [], [aj] \rangle, \langle [], [aa] \rangle, \ldots \}$$

$$[\![\text{happy}(\textit{pro})]\!]^g$$
$$= \{ \langle s, s' \rangle \mid s = s' \wedge s_\tau \in \text{happy} \}$$
$$= \{ \langle [j], [j] \rangle, \langle [aj], [aj] \rangle, \ldots \}$$

Cross-sentential binding ii

$$[\![\exists v[\text{hug}(v, \text{Annie})] \wedge \text{happy}(\textit{pro})]\!]^{g}$$
$$= \{ \langle [], [a\, j] \rangle, \ldots \}$$

- Notice that the last discourse referent pushed to the stack is j.
- Binding isn't going to work in the other direction...

Cross-sentential binding iii

$$[\![\text{happy}(pro)]\!]^g$$
$$= \{ \langle s, s' \rangle \mid s = s' \wedge s_\tau \in \text{happy} \}$$
$$= \{ \langle [j], [j] \rangle, .. \}$$

$$[\![\exists v[\text{hug(v,Annie)}]]\!]^g$$
$$= \{ \langle s, s' \rangle \mid \exists x[s' = \widehat{sax} \wedge \langle x, a \rangle \in \text{hug}] \}$$
$$= \{ \langle [], [aj] \rangle, \langle [], [aa] \rangle, \langle [j], [jaj] \rangle, \langle [j], [jaa] \rangle, \ldots \}$$

- It's obvious here that we're just going to get something with dynamic effects equivalent to *someone hugged Annie*.

Backwards binding

- Note the predictions out theory makes:

(93) Someone arrived. They$_{x}$ sat down.

(94) *They$_{x}$ sat down and someone arrived.

(95) Someone likes themselves$_{x}$.

(96) *They$_{x}$ like someone.

Trace binding

- If traces are interpreted as variables, rather than as *pro*, DyS predicts that traces cannot be dynamically bound.
- It's difficult to come up with examples to test this but maybe something like the following. Although note that this would typically be rules out syntactically.

(97) Does John know who left and t_x like Mary?

Features of DyS

- Some interesting features of DyS:
  - We don't require co-indexation between pronouns and their binders, only between quantifiers and their traces.
  - Coreference and binding are not syntactically distinguished.
  - Stack updates are always monotonic – we can always add individuals to the stack, but never reduce it.

Negation

- In DyS, additional operators can be added to achieve the same results as DPL.
- $[\![\neg\phi]\!]^g = \{ \langle s, s' \rangle \mid s = s' \wedge \neg\exists k : \langle s', k \rangle \in [\![\phi]\!]^g \}$
- $\neg\phi$ is interpreted as a test on stacks that *fail* to return a valid output when threaded into ϕ.

Negation ii

- We can see how negation is going to block dynamic binding
- Since every stack is a possible input to the existential statement, negation is going to return the empty set.

$$
\begin{aligned}
&[\![\neg \exists v[\text{happy}(v)]]\!]^g \\
&= \{ \langle s, s' \rangle \mid s = s' \wedge \neg k : \langle s', k \rangle \in \{ \langle t, t' \rangle \mid \exists x[t' = t\widehat{x} \wedge x \in \textsf{happy}] \} \} \\
&= \{ \langle s, s' \rangle \mid s = s' \wedge \neg k : \langle s', k \rangle \in \{ \langle [], [j] \rangle, \langle [a], [aj] \rangle, \langle [j], [jj] \rangle, \dots \} \} \\
&= \varnothing
\end{aligned}
$$

Exercise

- Define the rule for dynamic implication in DyS. Here is the rule from DPL to help you:
  - $[\![\phi \rightarrow \psi]\!] =$
    $\{\langle i, o\rangle \mid o = i \text{ and } \forall k : \langle o, k\rangle \in [\![\phi]\!] \Rightarrow \exists j : \langle k, j\rangle \in [\![\psi]\!]\}$
- Once you've done that, compute the meaning of the following.

(98) If someone hugs Annie, they$_x$ are happy.

- Again, assume a model where Jeff hugs Annie, everyone hugs themselves, nobody else hugs anybody else, and only Jeff is happy.

The novelty condition

- Heim proposed the *novely condition* to rule out cases like the following:

(99) Someone$^{\mathbf{1}}$ walked in. Someone$^{\mathbf{1}}$ sat down.

- In frameworks like DPL, the second conjunct "resets" the value of x_1, predicting that the sentence as a whole should just introduce a discourse referent that sat down.

- DyS doesn't have any need for the novely condition, since each existential statement adds a new discourse referent to the stack. Indices are irrelevant.

The novelty condition ii

$$[\![\exists v[\text{walkedIn}(v)]]\!]^g = \{ \langle t, t' \rangle \mid \exists x[t' = t\widehat{x} \wedge x \in \texttt{walkedIn}] \}$$

$$[\![\exists v[\text{satDown}(v)]]\!]^g = \{ \langle t, t' \rangle \mid \exists y[t' = t\widehat{y} \wedge y \in \texttt{satDown}] \}$$

$$[\![\exists v[\text{walkedIn}(v)] \wedge \exists v[\text{satDown}(v)]]\!]^g = \{ \langle t, t' \rangle \mid \exists xy[t' = \widehat{txy} \wedge x \in \texttt{walkedIn} \wedge y \in \texttt{satDown}] \}$$

The familiarity condition

- Heim also proposed the familiarity condition, to capture the fact that pronominals seem to presuppose the existence of a familiar discourse referent.

(100) (In an out-of-the-blue context) *He sat down.

- The way that Heim accomplishes this is by making assignments partial, and introducing a syntactic condition stating that pronouns should re-use an index that has already been introduced.

The familiarity condition

- DyS can capture something like the familiarity condition.
- If π is a unary predicate, then
  $[\![\pi(pro)]\!]^g = \{ \langle s, s' \rangle \mid s' = s \text{ and } s'_\tau \in [\![\pi]\!] \}$
  - Recall that formulas with pronouns are tests on a stack s', just in case s'_τ satisfies the predicate.
  - τ is a partial function that is only defined for stacks that have at least one discourse referent. If the input stack is empty, i.e., if there are no salient discourse referents, then τ is undefined.

Further directions

- Dynamic semantics was initially motivated by basic data concerning the interaction between scope and pronominal binding.
- But the empirical reach of dynamic semantics extends beyond this domain.
- Dynamic semantics is especially well-suited to analyzing phenomena which display a *dynamic signature* – an apparent sensitivity to linear order.

Tense

- Partee (1973) observed that their are parallels between tense and pronouns.
- Consider the following data:

(101) Pedro owns a donkey^{x}. He beats it$_{x}$.

(102) Yesterday, Pedro tried^{t} to kiss Juanita. She slapped$_{t \leq t'}$ him.

- The first sentence is a standard case of *donkey anaphora*. We are already equipped with the necessary tools to analyse this.
- The second sentence shows a similar phenomenon, but it the temporal domain – the past tense interpretation of the second clause is anaphorically dependent on the first. The sentence is conveys that Juanita slapped Pedro *right after* he tried to kiss her.

Tense ii

- The idea informally, is as follows. Action sentences are existentially quantified statements about events (Davidson, 1967). As well as more metaphysically conventional entities, we also have events in our domain.

(103) Pedro tried to kiss Juanita. – $\exists e_1[\text{tryToKiss}(p, j, e_1) \wedge e_1 \leq e]$

(104) She slapped him – $\exists e_2[\text{slapped}(x, y, e_2) \wedge e_2 \leq e_1]$

(105) Pedro tried to kiss Juanita and she slapped him.

$$\exists e_1[\text{tryToKiss}(p, j, e_1) \wedge e_1 \leq e] \wedge \exists e_2[\text{slapped}(x, y, e_2) \wedge e_2 \leq e_1]$$

- If we treat the existential quantifier over events as our familiar *dynamic* existential quantifier, then binding across conjuncts is predicted to be possible.

Donkey tense

- As well as cross-sentential tense anaphora, we can also create examples which exhibit the correlate of donkey anaphora in the temporal domain.

(106) Every farmer^{x} who owns a donkey^{y} t_x beat it$_y$.

(107) Whenever Pedro tried^{t} to kiss Juanita, she slapped$_{t \leq t'}$ him.

$$(\exists e_1[\text{tryToKiss}(p, j, e_1) \wedge e_1 \leq e]) \rightarrow (\exists e_2[\text{slap}(x, y, e_2) \wedge e_2 \leq e_1])$$

Presupposition

The phenomenon

- In this section we'll explore the other primary empirical motivation for a dynamic perspective on meaning.
- Certain expressions in natural language seem to carry preconditions.

(108) Mary stopped smoking

- Sounds fine in contexts where it's known that Mary used to smoke...
  - and is *true* iff she doesn't currently smoke.
- Sounds weird in contexts where it's known that Mary didn't used to smoke...
  - ...and that she doesn't currently smoke.
  - ...and that she does currently smoke.

Assertion vs. presupposition

- Generally speaking, certain predicates seem to have *two dimensions* to their meaning:
- The *assertive component*, which determines their truth/falsity.
    - for *stopped smoking*, the assertive component is *doesn't currently smoke*.
- The *presupposition*, which determines *whether* the sentence has a chance of being true or false in the first place.
    - for *stopped smoking*, the presupposition is *used to smoke*.
    - I'll write the presupposition of an expression P as $\Pi(P)$, and the assertion $\mathrm{A}(P)$.

Presupposition projection

- One way of distinguish the assertion vs. the presupposition, is that certain operators, such as *negation* are *presupposition holes* – in other words, negation affects the assertion of the sentence (it is negated), but the presupposition is not.

(109) Jeff didn't stop smoking

- $\mathbf{A}$: it's not the case that Jeff currently smokes.
- Π: Jeff used to smoke.
- Negation *projects* through negation.

Presupposition in a static setting

- In a static setting, the standard way of dealing with presupposition is to treat predicate meanings as *partial functions*.
- $\text{stoppedSmoking} = \lambda x : \text{usedToSmoke}(x) \,.\, \neg\text{smokesNow}(x)$
- This is a *function* from an individual x, which is defined if x used to smoke, and returns *true* iff x doesn't currently smoke.

Presupposition in a static setting

- The partial function $\mathbf{stoppedSmoking}$ is applied to an individual, such as $\mathbf{Jeff}$. Since $\mathbf{stoppedSmoking}$ is *partial*, it will return a truth value iff *Jeff used to smoke* is true, if not the sentence will be undefined.

$$[\lambda x : \text{usedToSmoke}(x) \,.\, \neg\text{smokesNow}(x)](\text{Jeff})$$

$$\text{defined iff } \text{usedToSmoke}(\text{Jeff}) = 1$$

$$= 1 \text{ iff } \neg\text{smokesNow}(\text{Jeff})$$

Presupposition in a static setting

- We can leave negation as a total function from truth values to true values.
- $\text{not} = \lambda t \,.\, \neg t$
- Functions are only defined, however, if their arguments are defined. If the argument ϕ of not is undefined, $\text{not}(\phi)$ is undefined.
- This basic story predicts projection through negation.

Presupposition in a static setting

$$\text{not}([\lambda x : \text{usedToSmoke}(x) \,.\, \neg\text{smokesNow}(x)](\text{Jeff}))$$

$$\text{defined iff usedToSmoke(Jeff)} = 1$$

$$= 1 \text{ iff smokesNow(Jeff)}$$

Trivalence

- Analyzing presuppositions using partial functions is largely *equivalent* to introducing a third truth value # to represent *undefined*.
- We can think of $\textbf{stoppedSmoking}$ as the following function.

$$\text{stoppedSmoking} = \lambda x\,.\begin{cases}\text{if } \neg\text{usedToSmoke}(x) \text{ then } \#\\ \text{if usedToSmoke}(x) \wedge \neg\text{smokes}(x) \text{ then } 1\\ \text{if usedToSmoke}(x) \wedge \text{smokes}(x) \text{ then } 0\end{cases}$$

Capturing projection

- The final step is to assign logical operators such as not *trivalent meanings*.

$$\text{not} = \lambda t \,.\, \begin{cases} t = \# & \# \\ t = 0 & 1 \\ t = 1 & 0 \end{cases}$$

Back to FOL

- We can recast the static account of presupposition projection in our FOL framework by formulating trivalent rules for atomic and complex formulae.
- If π is a unary predicate and α is a individual constant, then:
- $$[\![\pi(\alpha)]\!]^{M,g} = \begin{cases} \text{if } \Pi(\pi) = 0 \text{ then } \# \\ \text{if } \Pi(\pi) = 1 \text{ and } [\![\alpha]\!]^{M,g} \in [\![\pi]\!]^{M,g} \text{ then } 1 \\ \text{if } \Pi(\pi) = 1 \text{ and } [\![\alpha]\!]^{M,g} \notin [\![\pi]\!]^{M,g} \text{ then } 0 \end{cases}$$
- $$[\![\neg\phi]\!]^{M,g} = \begin{cases} [\![\phi]\!]^{M,g} = \# & \# \\ [\![\phi]\!]^{M,g} = 0 & 1 \\ [\![\phi]\!]^{M,g} = 1 & 0 \end{cases}$$

Exercise

- Formulate trivalent interpretation rules for the rest of the logical connectives. Come up with examples to determine what happens with #.
- $[\![\phi \wedge \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 1$ and $[\![\psi]\!]^{M,g} = 1$
- $[\![\phi \vee \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 1$ or $[\![\psi]\!]^{M,g} = 1$
- $[\![\phi \rightarrow \psi]\!]^{M,g} = 1$ unless $[\![\phi]\!]^{M,g} = 1$ and $[\![\psi]\!]^{M,g} = 0$
- $[\![\phi \leftrightarrow \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = [\![\psi]\!]^{M,g}$

A problem for static theories of presupposition

- Consider the following sentence:

(110) Jeff used to smoke and he stopped smoking.

- Do you think this *complex* sentence is presuppositional?

Using negation as a diagnostic

(111) It's not the case that Jeff used to smoke and he stopped smoking.

- This is true iff Jeff used to smoke, and doesn't currently smoke!
- We can conclude that this complex sentence *doesn't have a presupposition*

Trivalence?

- Let's check that our trivalent formulation of *and* isn't going to derive this.

$$[\![\phi \wedge \psi]\!]^{M,g} = \begin{cases} [\![\phi]\!]^{M,g} = \# \text{ or } [\![\psi]\!]^{M,g} = \# & \# \\ \text{else } [\![\phi]\!]^{M,g} = 0 \text{ or } [\![\psi]\!]^{M,g} = 0 & 0 \\ \text{otherwise} & 1 \end{cases}$$

- We predict, in fact that *Jeff used to smoke, and stopped smoking* should *inhert* the presupposition of *stopped smoking*.

- In other words, the presupposition of *stopped smoking* should project, but it doesn't

Back to Dynamic Semantics

(112) Jeff used to smoke, and he stopped smoking.

- Intuitively, the reason why this sentence lacks a presupposition, because the first conjunct asserts what is presupposed by the second sentence.
- Note that linear order seems to matter here.

(113) Jeff stopped smoking and he used to smoke.

- We find similar affects with implication.

(114) If John used to smoke, then he stopped smoking.

The Stalnakerian model of discourse

- Stalnaker's idea is that when we *assert* a sentence ϕ, the common ground C is *updated* with the content of ϕ.

(115) Jeff smokes $= C[\text{Jeff smokes}]$

- So far, we have been assuming that sentences are simply *true* or *false* (or maybe undefined). If sentences can only contribute truth-values, it's difficult to make sense of the idea that they can add information to the common ground.

- In order to make sense of this idea, we need to shift to a richer notion of content, so that we can distinguish between the informational content of sentences, regardless of whether they are true or false.

Possible worlds

- We will posit a new semantic object – a *possible world*.
- A possible world is simply a *way the world could be*, i.e., an index at which all of the facts are known.
- For example, there may be a world w_1, in which it's true that Jeff smokes, and a world w_2 where it's false that Jeff smokes.
- Philosophers argue frequently about the ontological nature of these beasts. For our purposes, they're a convenient mathematical construct for capturing the idea that we can reason about possibilties.

Possible worlds and propositions

- Possible worlds are indices at which *all of the facts are known*.
- So, at w_1, Jeff will have properties other than just smoking, maybe he's e.g., blonde. We will need to multiply our possible worlds to account for the range of possibilities:
    - w_1 =Jeff smokes and is blonde
    - w_2 =Jeff doesn't smoke and is blonde
    - w_3 =Jeff smokes and is brunette
    - w_4 = Jeff doesn't smoke and is brunette

Possible worlds and propositions ii

- Rather than thinking of a sentence such as *Jeff smokes* as expressing a *truth value*, we can think of it as expressing a *set of possible worlds*, namely, those possible worlds at which the sentence is true.
- Jeff smokes $= \{ w_1, w_3 \}$
- Jeff doesn't smoke $= \{ w_2, w_4 \}$
- Jeff is blonde $= \{ w_1, w_2 \}$
- Jeff is brunette $= \{ w_3, w_4 \}$

Enriching FOL

- Our *model* will now not just include a *domain* D, an *interpretation function* I, and a stock of *variables* V, but also a set of *possible worlds* W.

$$M = \langle D, I, V, W \rangle$$

- Our interpretation function I now takes an additional *possible world* argument, to reflect that fact that predicates can be true of different individuals in different worlds. We'll right $I_w(\alpha)$ to express the interpretation of α at w.

From Extensions to Intensions

- We can now reformulate our rule for atomic formulas, such that they express *sets of possible worlds* (intensions), rather than truth values (extensions).
- If π is a unary predicate and α is a individual constant, then:
  - $[\![\phi(\alpha)]\!]^{M,g,w'} = \{ w \mid w \in W \wedge [\![\alpha]\!]^{M,g,w} \in [\![\pi]\!]^{M,g,w} \}$
- This move is independently motivated, on the basis of verbs such as *believe*, etc.

Back to Stalnaker

- We can now make sense of Stalnaker's idea in a more precise way – the common ground can be thought of as just a *set of worlds*.
- For example, if we're not sure whether Jeff smokes or whether he's blonde or brunette, then the common ground $C = \{ w_1, w_2, w_3, w_4 \}$.
- Our rule for assertion states that, on asserting ϕ, we *intersect* ϕ with C.
- Jeff smokes $= C \cap \{ w \mid w \in W \wedge j \in I_w(\text{smoke}) \}$
- $= \{ w_1, w_2, w_3, w_4 \} \cap \{ w_1, w_3 \} = \{ w_1, w_3 \}$

Back to Stalnaker ii

- Contradictory sentences – those which are true in no possible world – simply express the empty set $\varnothing$.
- Asserting a contradiction will always result in $\varnothing$, since intersecting $\varnothing$ with any other set always returns $\varnothing$.

Back to Stalnaker iii

- On to Stalnaker's innovation – a conjunctive sentence such as *Jeff smokes and Jeff is blonde* involves *incrementally updating* the common ground C, in other words:
- Jeff smokes and Jeff is blonde = $\mathrm{C}[\text{Jeff smokes}][\text{Jeff is blonde}]$
- $C \cap \text{Jeff smokes} = \{ w_1, w_3 \}$
- $\{ w_1, w_3 \} \cap \text{Jeff is blonde} = \{ w_1, w_3 \} \cap \{ w_1, w_2 \} = \{ w_1 \}$
- The effect of incrementally updating C with first *Jeff smokes* followed by *Jeff is blonde* involves narrowing down C until we are just left with the world in which Jeff smokes and is blonde.

The flow of information

- In this way Stalnaker's framework involves modelling the flow of information over the course of a discourse.
- The more information we learn, the more we can narrow down C, which reflects how certain we are about how the world is.
- To see the similarities with the dynamic systems we've been looking at, suppose that we think of the meaning of an atomic formula as a *relation between sets of possible worlds*, i.e., an instruction for updating C.

$$\text{Jeff smokes} = \left\{ \langle \phi, \psi \rangle \mid \psi = \phi \cap \left\{ w \mid w \in W \wedge j \in I_w(\text{smokes}) \right\} \right\}$$

Stalnakerian conjunction

- Just like in dynamic semantics, we can think of conjunction as an iterative update.

$$
\begin{aligned}
& p \wedge q \\
& = \{ \langle \phi, \psi \rangle \mid \psi = (\phi \cap p) \cap q \}
\end{aligned}
$$

Stalnaker's proposal

- Let's consider our problematic example again.

(116) Jeff used to smoke and Jeff stopped smoking.

- Using our dynamic interpretation scheme, this will be interpreted as:

$$\left\{ \langle \phi, \psi \rangle \;\middle|\; \psi = \begin{array}{l} (\phi \cap \{\, w \mid j \in I_w(\text{smoked}) \,\}) \\ \quad \cap \{\, w' \mid j \in I_{w'}(\text{stoppedSmoking}) \,\} \end{array} \right\}$$

$$\Pi(\text{Jeff stopped smoking}) = \text{John smoked}$$

Stalnaker's proposal ii

- Since $\Pi(\text{stoppedSmoking})$ is entailed by $\text{A}(\text{smoked})$, this incremental update will *always* be defined.
- Therefore, our interpretation is equivalent to:

$$\left\{ \langle \phi, \psi \rangle \;\middle|\; \psi = \begin{array}{c} (\phi \cap \{\, w \mid j \in I_w(\text{smoked}) \,\}) \\ \cap \{\, w' \mid \neg(j \in I_{w'}(\text{smoke})) \,\} \end{array} \right\}$$

- which is presuppositionless! The presupposition is *locally satisfied* by the first conjunct.

Function composition

- Let's try to reformulate this idea in terms of functions to make sure we understand the basic idea.

$$P \circ Q = \lambda\, x\, .\, P\, (Q\, x)$$

Stalnaker's proposal (with functions)

$$P = \lambda p\,.[\lambda w\,.\,j \in \textsf{smoked}] \wedge p$$

$$Q = \lambda p\,.[\lambda w : j \in \textsf{smoked}\,.\,\neg(j \in \textsf{smokes})] \wedge p$$

$$P \text{ and } Q = \lambda p\,.\,Q \circ P \circ p$$
$$= \lambda p\,.\,[\lambda\, w\,.\,j \in \textsf{smoked} \wedge \neg(j \in \textsf{smokes})] \wedge p$$

$$Q_\Pi \circ P \equiv Q \circ P \text{ iff } P \Rightarrow \Pi(Q)$$

Other operators

- To a large extent, structural conditions on local satisfaction of presuppositions are identical to those of dynamic binding.

(117) If John used to smoke$^{\Pi}$ then he stopped smoking$_{\Pi}$

(118) If a man^{x} arrived, then he$_{x}$ sat down.

(119) Every man who used to smoke$^{\Pi}$ has stopped smoking$_{\Pi}$

(120) Every man who met a professor^{x} admired her$_{x}$.

Accessibility

- The same operators that block dynamic binding block local satisfaction of a presupposition.

(121) *John didn't ever smoke$^{\Pi}$ and he stopped smoking$_{\Pi}$.

(122) *Nobody left and he$_{x}$ sat down.

(123) *If Mary used to smoke$^{\Pi}$ then John did too. She stopped smoking$_{\Pi}$.

(124) *If someoned arrived, they$_{x}$ sat down. They$_{x}$ left soon afterwards.

The underlying dynamics

- Just as we analyze the dynamic nature of anaphora by treating sentence meanings as relations between assignments (or stacks if you prefer), we can analyse the dynamic nature of presupposition satisfaction by treating sentence meanings as relations between propositions.
- Ultimately, we want to have a single logical system which provides a unification of both ideas.
- For such a system, see for example Rothschild (2017).
- I won't try to lay out the formal details here.

A problem

- A brief note: a dynamic theory of presupposition satisfaction inherits much of the same problems as a dynamic approach to anaphora.

(125) Either nobody^{x} is coming, or they$_{x}$ are late.

(126) Either Jane hasn't ever smoked$^{\Pi}$, or she stopped smoking$_{\Pi}$.

- The local antecedent to both the anaphor and the presupposition should be rendered inaccessible – but it isn't.

Overview

- In this course, I've taken you from the most bare-bones logical system necessary for analysing a fragment of natural language – namely L_1, to a dynamic logical system **DyS**, which accounts for the behaviour of anaphora and binding in a wide variety of environments.
- To do this, we've incrementally built on L_1 adding new features as we encountered new data that motivated them.
- Throughout, we've maintained the core tenets of formal semantics – namely, that we can analyze meaning in natural language using a rigorous procedure for mapping expressions in a logical language to *meanings*.
    - In L_1, meanings were *truth values*.
    - In FOL, meanings were *sets of assignments*.
    - In DPL and **DyS**, meanings were *relations between assignments (or stacks)*

Overview ii

- The methods and formal tools available allow us to build on the results of our earlier theories *without throwing the baby out with the bathwater*.
- In the final class, we've given an overview of phenomena that motivate the dynamic perspective beyond just anaphora and binding.
- Even if you don't plan on being a formal semanticists, hopefully we've managed to convey the advantages to analyzing natural language with the level of rigour and precision that formal tools afford us.

Bibliography i

- Charlow, Simon. 2014. *On the semantics of exceptional scope*. New York University dissertation.
- Dekker, Paul. 1994. Predicate logic with anaphora. *Semantics and Linguistic Theory* 4(0). 79–95.
- Groenendijk, Jeroen & Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy* 14(1). 39–100.
- Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar*. (Blackwell textbooks in linguistics 13). Malden, MA: Blackwell. 324 pp.
- Jacobson, Pauline I. 2014. *Compositional semantics: an introduction to the syntax/semantics interface*. (Oxford textbooks in linguistics). New York, NY: Oxford University Press. 427 pp.
- Karttunen, Lauri. 1976. Discourse referents. In J. D. McCawley (ed.), *Syntax and semantics vol. 7*, 363–386. Academic Press.


Rothschild, Daniel. 2017. A trivalent approach to anaphora and presupposition. unpublished manuscript.