Hyperintensional Dynamic Semantics

Toward a Higher-Order Theory of Presupposition

Carl Pollard

Department of Linguistics
Ohio State University

November 8, 2010

Dynamic Categorial Grammar (DyCG)

- An interdisciplinary seminar at Ohio State University
- Initiated in Spring 2009 by Scott Martin, Carl Pollard, Craige Roberts, and Elizabeth Smith
- Goal: an integrated theory of NL syntax, semantics, and pragmatics which is
  - formally explicit
  - computationally implemented
  - pedagogically sound (comprehensible to linguists)
  - equipped to handle presupposition and other kinds of projective meaning

DyCG Integrates Three Research Traditions

- the *curryesque* tradition in categorial grammar
- the *hyperintensional* tradition in sentence semantics
- the *dynamic* tradition in discourse semantics

Curryesque Categorial Grammar (1/2)

- Curry's (1961) grammar architecture: *tectogrammar* (abstract syntax) mediates between *phenogrammar* (concrete syntax) and semantics

Curryesque Categorial Grammar (1/2)

- Curry's (1961) grammar architecture: *tectogrammar* (abstract syntax) mediates between *phenogrammar* (concrete syntax) and semantics
- Cresswell's (1973) λ-*categorial grammars*
  - first use of λ-abstraction in *syntactic* terms
  - use of *tupling* in syntactic terms anticipates later use of string terms at pheno level

Curryesque Categorial Grammar (2/2)

- Oehrle (1994):
  - linear logic for tecto types, λ-calculi for pheno and semantics
  - string concatenation as a term-forming operation at the pheno level
  - Montague's 'quantifier lowering' implemented via β-reduction at the pheno level

Curryesque Categorial Grammar (2/2)

- Oehrle (1994):
  - linear logic for tecto types, λ-calculi for pheno and semantics
  - string concatenation as a term-forming operation at the pheno level
  - Montague's 'quantifier lowering' implemented via β-reduction at the pheno level
- de Groote's (2001) ACG, Pollard's (2001) HOG
  - term calculi at all three levels (pheno, tecto, semantics)
  - categorical functors from tecto to pheno and to semantics

Curryesque Categorial Grammar (2/2)

- Oehrle (1994):
  - linear logic for tecto types, λ-calculi for pheno and semantics
  - string concatenation as a term-forming operation at the pheno level
  - Montague's 'quantifier lowering' implemented via β-reduction at the pheno level
- de Groote's (2001) ACG, Pollard's (2001) HOG
  - term calculi at all three levels (pheno, tecto, semantics)
  - categorical functors from tecto to pheno and to semantics
- Muskens' (2001) λ-grammar: 'overt movement traces' implemented as 'lowering' of null string at pheno level

Hyperintensional Semantics

- Finer-grained alternatives to Montague semantics (Thomason 1980, Pollard 2004, Muskens 2007)
- Does everything Montague semantics does, but avoids some of its foundational problems.

Hyperintensional Semantics

- Finer-grained alternatives to Montague semantics (Thomason 1980, Pollard 2004, Muskens 2007)
- Does everything Montague semantics does, but avoids some of its foundational problems.
- Propositions (sentence meanings) are a *basic* type, rather than defined as sets of worlds.
- The entailment relation on propositions is not antisymmetric, so truth-conditionally equivalent propositions need not be equal.

Hyperintensional Semantics

- Finer-grained alternatives to Montague semantics (Thomason 1980, Pollard 2004, Muskens 2007)
- Does everything Montague semantics does, but avoids some of its foundational problems.
- Propositions (sentence meanings) are a *basic* type, rather than defined as sets of worlds.
- The entailment relation on propositions is not antisymmetric, so truth-conditionally equivalent propositions need not be equal.
- The Montagovian menagerie (worlds, intensions, extensions) are all definable, but NL grammars never need to make reference to them.

Dynamic Semantics: Introduction

- Pioneered by Kamp (1981), Heim (1982), Barwise (1986), Rooth (1986).
- Handles phenomena not handled by 'static' semantics (no matter whether Montagovian or hyperintensional):
  - novelty condition on indefinites
  - cross-sentential anaphora
  - donkey anaphora

Dynamic Semantics: Basic Notions

- Sentence meanings are relations (or partial functions) between *contexts*.
- Sentence conjunction is interpreted as composition.

Dynamic Semantics: Basic Notions

- Sentence meanings are relations (or partial functions) between *contexts*.
- Sentence conjunction is interpreted as composition.
- Uttering an indefinite (*a donkey*) introduces a new *discourse referent* (DR) into the context.

Dynamic Semantics: Basic Notions

- Sentence meanings are relations (or partial functions) between *contexts*.
- Sentence conjunction is interpreted as composition.
- Uttering an indefinite (*a donkey*) introduces a new *discourse referent* (DR) into the context.
- DR's are abstract objects that mediate between the discourse and the individuals the discourse is about.

Dynamic Semantics: Basic Notions

- Sentence meanings are relations (or partial functions) between *contexts*.
- Sentence conjunction is interpreted as composition.
- Uttering an indefinite (*a donkey*) introduces a new *discourse referent* (DR) into the context.
- DR's are abstract objects that mediate between the discourse and the individuals the discourse is about.
- DR's (not individuals) are the 'antecedents' of definite expressions (such as pronouns, names, and definite descriptions).

Dynamic Semantics: Drawbacks

- Not much agreement about the mathematical/logical foundations.
- Not much agreement about how to model contexts.
- Unclear how to model *presuppositions*, the conditions on contexts that must hold for an utterance to be felicitous.

Dynamic Semantics Meets Higher Order Logic

- Muskens (1996) and de Groote (2006) have each made explicit proposals for extending Montague semantics to handle the 'classic' dynamic phenomena.
- No formal devices beyond ordinary classical HOL

Dynamic Semantics Meets Higher Order Logic

- Muskens (1996) and de Groote (2006) have each made explicit proposals for extending Montague semantics to handle the 'classic' dynamic phenomena.
- No formal devices beyond ordinary classical HOL
- But these proposals are themselves problematic:
  - Muskens treats anaphora as nondeterministic.

Dynamic Semantics Meets Higher Order Logic

- Muskens (1996) and de Groote (2006) have each made explicit proposals for extending Montague semantics to handle the 'classic' dynamic phenomena.
- No formal devices beyond ordinary classical HOL
- But these proposals are themselves problematic:
  - Muskens treats anaphora as nondeterministic.
  - de Groote treats anaphora as *too* deterministic (oracular).

Dynamic Semantics Meets Higher Order Logic

- Muskens (1996) and de Groote (2006) have each made explicit proposals for extending Montague semantics to handle the 'classic' dynamic phenomena.
- No formal devices beyond ordinary classical HOL
- But these proposals are themselves problematic:
  - Muskens treats anaphora as nondeterministic.
  - de Groote treats anaphora as *too* deterministic (oracular).
  - Neither extends straightforwardly to presuppositions other than definite pronominal anaphora.

This Talk

- sketches ongoing efforts to extend hyperintensional semantics to dynamic phenomena.
- We borrow ideas from both Muskens and de Groote, but employ a richer notion of context.

This Talk

- sketches ongoing efforts to extend hyperintensional semantics to dynamic phenomena.
- We borrow ideas from both Muskens and de Groote, but employ a richer notion of context.
- Our approach for modelling contexts is adapted from Roberts (1996, 2004), which in turn draw inspiration from Lewis and Stalnaker.

This Talk

- sketches ongoing efforts to extend hyperintensional semantics to dynamic phenomena.
- We borrow ideas from both Muskens and de Groote, but employ a richer notion of context.
- Our approach for modelling contexts is adapted from Roberts (1996, 2004), which in turn draw inspiration from Lewis and Stalnaker.
- We consider a sampling of presuppositional phenomena:
  - pronouns
  - definite descriptions
  - projection (e.g. through negation)
  - factivity
  - independence of antecedent-of-conditional

Technical Preliminaries: Higher Order Logic

We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types t (truth values) and ω (natural numbers)
- the usual cartesian-closed type constructors 1 (unit), $\times$ (product), and $\rightarrow$ (exponential)

Technical Preliminaries: Higher Order Logic

We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types t (truth values) and ω (natural numbers)
- the usual cartesian-closed type constructors 1 (unit), $\times$ (product), and $\rightarrow$ (exponential)
- separation subtyping:
  - for any type A and formula $\varphi[x]$ with at most x of type A free, there is a type $\{x \in A \mid \varphi[x]\}$.

Technical Preliminaries: Higher Order Logic

We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types t (truth values) and ω (natural numbers)
- the usual cartesian-closed type constructors 1 (unit), $\times$ (product), and $\rightarrow$ (exponential)
- separation subtyping:
  - for any type A and formula $\varphi[x]$ with at most x of type A free, there is a type $\{x \in A \mid \varphi[x]\}$.
  - In a set-theoretic interpretation I, this is interpreted as the subset of $I(A)$ whose characteristic function is $I(\lambda_x.\varphi[x])$.

Technical Preliminaries: Higher Order Logic

We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types t (truth values) and ω (natural numbers)
- the usual cartesian-closed type constructors 1 (unit), $\times$ (product), and $\rightarrow$ (exponential)
- separation subtyping:
  - for any type A and formula $\varphi[x]$ with at most x of type A free, there is a type $\{x \in A \mid \varphi[x]\}$.
  - In a set-theoretic interpretation I, this is interpreted as the subset of $I(A)$ whose characteristic function is $I(\lambda_x.\varphi[x])$.
- Countable disjoint union types (dependent coproducts parametrized by ω), written $\coprod_{n:\omega} A_n$.

Technical Preliminaries: Some Defined Types

- $\omega_n =_{\text{def}} \{i \in \omega \mid i < n\}$: the first n natural numbers

Technical Preliminaries: Some Defined Types

- $\omega_n =_{\text{def}} \{i \in \omega \mid i < n\}$: the first n natural numbers
- $\text{Rel}_{A,B} =_{\text{def}} A \to B \to t$: relations from A to B

Technical Preliminaries: Some Defined Types

- $\omega_n =_{\text{def}} \{i \in \omega \mid i < n\}$: the first n natural numbers
- $\text{Rel}_{A,B} =_{\text{def}} A \to B \to t$: relations from A to B
- $A \rightharpoonup B$ (subtype of $\text{Rel}_{A,B}$): partial functions from A to B

Technical Preliminaries: Some Defined Types

- $\omega_n =_{\text{def}} \{i \in \omega \mid i < n\}$: the first n natural numbers
- $\text{Rel}_{A,B} =_{\text{def}} A \to B \to t$: relations from A to B
- $A \rightharpoonup B$ (subtype of $\text{Rel}_{A,B}$): partial functions from A to B
- Preord_A (subtype of $\text{Rel}_{A,A}$): preorders on A

Technical Preliminaries: Some Defined Types

- $\omega_n =_{\text{def}} \{i \in \omega \mid i < n\}$: the first n natural numbers
- $\text{Rel}_{A,B} =_{\text{def}} A \to B \to t$: relations from A to B
- $A \rightharpoonup B$ (subtype of $\text{Rel}_{A,B}$): partial functions from A to B
- Preord_A (subtype of $\text{Rel}_{A,A}$): preorders on A
- Notational abuse: we write $\lambda_{x \mid \varphi[x]}.M[x]$ instead of $\lambda_y.M[y]$ when the type of y is a subtype $\{x \in T \mid \varphi[x]\}$.

Hyperintensional Semantics Basics

- Basic types e (entities) and p (propositions)

Hyperintensional Semantics Basics

- Basic types e (entities) and p (propositions)
- The **entailment** relation entails on propositions is axiomatized as a boolean preorder with the following operations:
  - and : meet
  - or : join
  - not: complement
  - implies : relative complement
  - true: top
  - false: bottom

Hyperintensional Semantics Basics

- Basic types e (entities) and p (propositions)
- The **entailment** relation entails on propositions is axiomatized as a boolean preorder with the following operations:
  - and : meet
  - or : join
  - not: complement
  - implies : relative complement
  - true: top
  - false: bottom
- But entailment is *not* assumed to be antisymmetric.

Hyperintensional Semantics Basics

- Basic types e (entities) and p (propositions)
- The **entailment** relation entails on propositions is axiomatized as a boolean preorder with the following operations:
  - and : meet
  - or : join
  - not: complement
  - implies : relative complement
  - true: top
  - false: bottom
- But entailment is *not* assumed to be antisymmetric.
- The type w of worlds is *defined* as a subtype of $p \to t$ (the ultrafilters).

Hyperintensional, Intensional, and Extensional Types

- The **hyperintensional** types are p, e, and types obtained from these using the type constructors.

Hyperintensional, Intensional, and Extensional Types

- The **hyperintensional** types are p, e, and types obtained from these using the type constructors.
- For each hyperintensional type A, the corresponding **extensional** type $\mathrm{Ext}(A)$ is defined as follows:
  - $\mathrm{Ext}(\mathrm{p}) = \mathrm{t}$

Hyperintensional, Intensional, and Extensional Types

- The **hyperintensional** types are p, e, and types obtained from these using the type constructors.
- For each hyperintensional type A, the corresponding **extensional** type $\mathrm{Ext}(A)$ is defined as follows:
  - $\mathrm{Ext}(\mathrm{p}) = \mathrm{t}$
  - $\mathrm{Ext}(\mathrm{e}) = \mathrm{e}$

Hyperintensional, Intensional, and Extensional Types

- The **hyperintensional** types are p, e, and types obtained from these using the type constructors.
- For each hyperintensional type A, the corresponding **extensional** type $\mathrm{Ext}(A)$ is defined as follows:
  - $\mathrm{Ext}(\mathrm{p}) = \mathrm{t}$
  - $\mathrm{Ext}(\mathrm{e}) = \mathrm{e}$
  - $\mathrm{Ext}(B \to C) = B \to \mathrm{Ext}(C)$

Hyperintensional, Intensional, and Extensional Types

- The **hyperintensional** types are p, e, and types obtained from these using the type constructors.
- For each hyperintensional type A, the corresponding **extensional** type $\mathrm{Ext}(A)$ is defined as follows:
  - $\mathrm{Ext}(\mathrm{p}) = \mathrm{t}$
  - $\mathrm{Ext}(\mathrm{e}) = \mathrm{e}$
  - $\mathrm{Ext}(B \to C) = B \to \mathrm{Ext}(C)$
- and the corresponding **intensional** type $\mathrm{Int}(A)$ is defined as $\mathrm{w} \to \mathrm{Ext}(A)$.

Hyperintensional, Intensional, and Extensional Types

- The **hyperintensional** types are p, e, and types obtained from these using the type constructors.
- For each hyperintensional type A, the corresponding **extensional** type $\mathrm{Ext}(A)$ is defined as follows:
  - $\mathrm{Ext}(\mathrm{p}) = \mathrm{t}$
  - $\mathrm{Ext}(\mathrm{e}) = \mathrm{e}$
  - $\mathrm{Ext}(B \to C) = B \to \mathrm{Ext}(C)$
- and the corresponding **intensional** type $\mathrm{Int}(A)$ is defined as $\mathrm{w} \to \mathrm{Ext}(A)$.
- So there are intensions. But meanings are hyperintensions, not intensions.

Hyperintensions, Intensions, and Extensions

- If $a : A$ is a hyperintension and w a world, the **extension** of a at w, written $a@w : \mathrm{Ext}(A)$, is defined to be:
  - $(w\ a) : \mathrm{t}$ if $A = \mathrm{p}$

Hyperintensions, Intensions, and Extensions

- If $a : A$ is a hyperintension and w a world, the **extension** of a at w, written $a@w : \mathrm{Ext}(A)$, is defined to be:
  - $(w\ a) : \mathrm{t}$ if $A = \mathrm{p}$
  - $a : \mathrm{e}$ if $A = \mathrm{e}$

Hyperintensions, Intensions, and Extensions

- If $a : A$ is a hyperintension and w a world, the **extension** of a at w, written $a@w : \mathrm{Ext}(A)$, is defined to be:
  - $(w\ a) : \mathrm{t}$ if $A = \mathrm{p}$
  - $a : \mathrm{e}$ if $A = \mathrm{e}$
  - $\lambda_x.(a\ x)@w : B \to \mathrm{Ext}(C)$ if $A = B \to C$

Hyperintensions, Intensions, and Extensions

- If $a : A$ is a hyperintension and w a world, the **extension** of a at w, written $a@w : \mathrm{Ext}(A)$, is defined to be:
  - $(w\ a) : \mathrm{t}$ if $A = \mathrm{p}$
  - $a : \mathrm{e}$ if $A = \mathrm{e}$
  - $\lambda_x.(a\ x)@w : B \to \mathrm{Ext}(C)$ if $A = B \to C$
- For each hyperintensional type A, the **intensionalizer** function is $\mathbf{int}_A =_{\mathrm{def}} \lambda_{wx}.x@w : A \to \mathrm{Int}(A)$ and for each $a : A$, $(\mathbf{int}\ a)$ is called the **intension corresponding to** a.

Hyperintensions, Intensions, and Extensions

- If $a : A$ is a hyperintension and w a world, the **extension** of a at w, written $a@w : \mathrm{Ext}(A)$, is defined to be:
  - $(w\ a) : \mathrm{t}$ if $A = \mathrm{p}$
  - $a : \mathrm{e}$ if $A = \mathrm{e}$
  - $\lambda_x.(a\ x)@w : B \to \mathrm{Ext}(C)$ if $A = B \to C$
- For each hyperintensional type A, the **intensionalizer** function is $\mathbf{int}_A =_{\mathrm{def}} \lambda_{wx}.x@w : A \to \mathrm{Int}(A)$ and for each $a : A$, $(\mathbf{int}\ a)$ is called the **intension corresponding to** a.
- $\mathbf{int}_{\mathrm{p}} : \mathrm{p} \to \mathrm{w} \to \mathrm{t}$ is the Stone dual mapping that maps each proposition to the set of worlds which contain it.

Hyperintensions, Intensions, and Extensions

- If $a : A$ is a hyperintension and w a world, the **extension** of a at w, written $a@w : \mathrm{Ext}(A)$, is defined to be:
  - $(w\ a) : \mathrm{t}$ if $A = \mathrm{p}$
  - $a : \mathrm{e}$ if $A = \mathrm{e}$
  - $\lambda_x.(a\ x)@w : B \to \mathrm{Ext}(C)$ if $A = B \to C$
- For each hyperintensional type A, the **intensionalizer** function is $\mathbf{int}_A =_{\mathrm{def}} \lambda_{wx}.x@w : A \to \mathrm{Int}(A)$ and for each $a : A$, $(\mathbf{int}\ a)$ is called the **intension corresponding to** a.
- $\mathbf{int}_{\mathrm{p}} : \mathrm{p} \to \mathrm{w} \to \mathrm{t}$ is the Stone dual mapping that maps each proposition to the set of worlds which contain it.
- Since entailment is not antisymmetric, $\mathbf{int}_{\mathrm{p}}$ is many-to-one.

Hyperintensions, Intensions, and Extensions

- If $a : A$ is a hyperintension and w a world, the **extension** of a at w, written $a@w : \mathrm{Ext}(A)$, is defined to be:
  - $(w\ a) : \mathrm{t}$ if $A = \mathrm{p}$
  - $a : \mathrm{e}$ if $A = \mathrm{e}$
  - $\lambda_x.(a\ x)@w : B \to \mathrm{Ext}(C)$ if $A = B \to C$
- For each hyperintensional type A, the **intensionalizer** function is $\mathbf{int}_A =_{\mathrm{def}} \lambda_{wx}.x@w : A \to \mathrm{Int}(A)$ and for each $a : A$, $(\mathbf{int}\ a)$ is called the **intension corresponding to** a.
- $\mathbf{int}_{\mathrm{p}} : \mathrm{p} \to \mathrm{w} \to \mathrm{t}$ is the Stone dual mapping that maps each proposition to the set of worlds which contain it.
- Since entailment is not antisymmetric, $\mathbf{int}_{\mathrm{p}}$ is many-to-one.
- That's why hyperintensional semantics is more fine-grained than Montague semantics.

Hyperintensional Quantifiers

- $\mathsf{exists} : (\mathrm{e} \to \mathrm{p}) \to \mathrm{p}$
  Axiom: $\vdash \forall_{P:\mathrm{e}\to\mathrm{p}} \forall_{w:\mathrm{w}} . (\mathsf{exists}\ P)@w = \exists_x . (P\ x)@w$

Hyperintensional Quantifiers

- $\mathsf{exists} : (\mathrm{e} \to \mathrm{p}) \to \mathrm{p}$
  Axiom: $\vdash \forall_{P:\mathrm{e}\to\mathrm{p}} \forall_{w:\mathrm{w}} . (\mathsf{exists}\ P)@w = \exists_x . (P\ x)@w$
- $\mathsf{forall} : (\mathrm{e} \to \mathrm{p}) \to \mathrm{p}$
  Axiom: $\vdash \forall_{P:\mathrm{e}\to\mathrm{p}} \forall_{w:\mathrm{w}} . (\mathsf{forall}\ P)@w = \forall_x . (P\ x)@w$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \rightarrow \mathrm{p}$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \rightarrow \mathrm{p}$

$bray \rightsquigarrow \mathsf{bray} : \mathrm{e} \rightarrow \mathrm{p}$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \to \mathrm{p}$

$bray \rightsquigarrow \mathsf{bray} : \mathrm{e} \to \mathrm{p}$

$own \rightsquigarrow \mathsf{own} : \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \to \mathrm{p}$

$bray \rightsquigarrow \mathsf{bray} : \mathrm{e} \to \mathrm{p}$

$own \rightsquigarrow \mathsf{own} : \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

$it_{\mathsf{dummy}} \rightsquigarrow * : 1$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \to \mathrm{p}$

$bray \rightsquigarrow \mathsf{bray} : \mathrm{e} \to \mathrm{p}$

$own \rightsquigarrow \mathsf{own} : \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

$it_{\mathsf{dummy}} \rightsquigarrow * : 1$

$rain \rightsquigarrow \lambda_u.\mathsf{rain} : 1 \to \mathrm{p}$ where $\mathsf{rain} : \mathrm{p}$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \to \mathrm{p}$

$bray \rightsquigarrow \mathsf{bray} : \mathrm{e} \to \mathrm{p}$

$own \rightsquigarrow \mathsf{own} : \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

$it_{\mathsf{dummy}} \rightsquigarrow * : 1$

$rain \rightsquigarrow \lambda_u.\mathsf{rain} : 1 \to \mathrm{p}$ where $\mathsf{rain} : \mathrm{p}$

$suck \rightsquigarrow \lambda_{pu}.\mathsf{suck}\, p : \mathrm{p} \to 1 \to \mathrm{p}$ where $\mathsf{suck} : \mathrm{p} \to \mathrm{p}$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \to \mathrm{p}$

$bray \rightsquigarrow \mathsf{bray} : \mathrm{e} \to \mathrm{p}$

$own \rightsquigarrow \mathsf{own} : \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

$it_{\mathsf{dummy}} \rightsquigarrow * : 1$

$rain \rightsquigarrow \lambda_u.\mathsf{rain} : 1 \to \mathrm{p}$ where $\mathsf{rain} : \mathrm{p}$

$suck \rightsquigarrow \lambda_{pu}.\mathsf{suck}\, p : \mathrm{p} \to 1 \to \mathrm{p}$ where $\mathsf{suck} : \mathrm{p} \to \mathrm{p}$

$no\ way \rightsquigarrow \mathsf{not} : \mathrm{p} \to \mathrm{p}$

Some Word Translations

$Pedro \rightsquigarrow \mathsf{pedro} : \mathrm{e}$

$donkey \rightsquigarrow \mathsf{donkey} : \mathrm{e} \to \mathrm{p}$

$bray \rightsquigarrow \mathsf{bray} : \mathrm{e} \to \mathrm{p}$

$own \rightsquigarrow \mathsf{own} : \mathrm{e} \to \mathrm{e} \to \mathrm{p}$

$it_{\mathsf{dummy}} \rightsquigarrow * : 1$

$rain \rightsquigarrow \lambda_u.\mathsf{rain} : 1 \to \mathrm{p}$ where $\mathsf{rain} : \mathrm{p}$

$suck \rightsquigarrow \lambda_{pu}.\mathsf{suck}\, p : \mathrm{p} \to 1 \to \mathrm{p}$ where $\mathsf{suck} : \mathrm{p} \to \mathrm{p}$

$no\ way \rightsquigarrow \mathsf{not} : \mathrm{p} \to \mathrm{p}$

$a \rightsquigarrow \lambda_{P,Q}.(\exists\, \lambda_x.(P\, x)\ \mathsf{and}\ (Q\, x)) : (\mathrm{e} \to \mathrm{p}) \to (\mathrm{e} \to \mathrm{p}) \to \mathrm{p}$

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Pedro owns Chiquita $\rightsquigarrow$ own chiquita pedro

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Pedro owns Chiquita $\rightsquigarrow$ own chiquita pedro

It rains $\rightsquigarrow$ $(\lambda_u.\mathsf{rain}) * = \mathsf{rain}$

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Pedro owns Chiquita $\rightsquigarrow$ own chiquita pedro

It rains $\rightsquigarrow$ $(\lambda_u.\mathsf{rain}) * = \mathsf{rain}$

It sucks that it rains $\rightsquigarrow$ suck rain

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Pedro owns Chiquita $\rightsquigarrow$ own chiquita pedro

It rains $\rightsquigarrow$ $(\lambda_u.\text{rain}) * =$ rain

It sucks that it rains $\rightsquigarrow$ suck rain

Pedro owns a donkey $\rightsquigarrow$ exists $\lambda_x.(\text{donkey } x)$ and $(\text{own } x \text{ pedro})$

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Pedro owns Chiquita $\rightsquigarrow$ own chiquita pedro

It rains $\rightsquigarrow$ $(\lambda_u.\text{rain}) * = \text{rain}$

It sucks that it rains $\rightsquigarrow$ suck rain

Pedro owns a donkey $\rightsquigarrow$ exists $\lambda_x.(\text{donkey } x)$ and $(\text{own } x \text{ pedro})$

- Remember that these meanings are propositions (type p), not truth values (type t) or Carnap/Montague-style intensions (type w $\rightarrow$ t).

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Pedro owns Chiquita $\rightsquigarrow$ own chiquita pedro

It rains $\rightsquigarrow$ $(\lambda_u.\text{rain}) * = \text{rain}$

It sucks that it rains $\rightsquigarrow$ suck rain

Pedro owns a donkey $\rightsquigarrow$ exists $\lambda_x.(\text{donkey } x)$ and $(\text{own } x \text{ pedro})$

- Remember that these meanings are propositions (type p), not truth values (type t) or Carnap/Montague-style intensions (type w $\rightarrow$ t).
- If you want to know whether one of them is true at a world w, you just have to see whether it is a member of w.

Some Sentence Translations (Given by the Grammar)

Chiquita brays $\rightsquigarrow$ bray chiquita

No way Chiquita brays $\rightsquigarrow$ not (bray chiquita)

Pedro owns Chiquita $\rightsquigarrow$ own chiquita pedro

It rains $\rightsquigarrow$ $(\lambda_u.\text{rain}) * = \text{rain}$

It sucks that it rains $\rightsquigarrow$ suck rain

Pedro owns a donkey $\rightsquigarrow$ exists $\lambda_x.(\text{donkey } x)$ and $(\text{own } x \text{ pedro})$

- Remember that these meanings are propositions (type p), not truth values (type t) or Carnap/Montague-style intensions (type w $\rightarrow$ t).
- If you want to know whether one of them is true at a world w, you just have to see whether it is a member of w.
- That's a question about the world, not a linguistic issue.

Introducing Contexts

Synthesizing suggestions of Lewis, Stalnaker, Heim, and Roberts, we take contexts to consist minimally of the following things:

Introducing Contexts

Synthesizing suggestions of Lewis, Stalnaker, Heim, and Roberts, we take contexts to consist minimally of the following things:

- a list of **discourse referents** (DRs) and an **anchor** function mapping the DRs to entities

Introducing Contexts

Synthesizing suggestions of Lewis, Stalnaker, Heim, and Roberts, we take contexts to consist minimally of the following things:

- a list of **discourse referents** (DRs) and an **anchor** function mapping the DRs to entities
  We prefer Barwise's term **anchor** to the usual term **assignment** because DRs are not object-language variables, they are a type of abstract semantic object.

Introducing Contexts

Synthesizing suggestions of Lewis, Stalnaker, Heim, and Roberts, we take contexts to consist minimally of the following things:

- a list of **discourse referents** (DRs) and an **anchor** function mapping the DRs to entities
  We prefer Barwise's term **anchor** to the usual term **assignment** because DRs are not object-language variables, they are a type of abstract semantic object.
- a **common ground** (CG), the conjunction of the propositions taken to be mutually accepted

Introducing Contexts

Synthesizing suggestions of Lewis, Stalnaker, Heim, and Roberts, we take contexts to consist minimally of the following things:

- a list of **discourse referents** (DRs) and an **anchor** function mapping the DRs to entities
  We prefer Barwise's term **anchor** to the usual term **assignment** because DRs are not object-language variables, they are a type of abstract semantic object.
- a **common ground** (CG), the conjunction of the propositions taken to be mutually accepted
- a notion of relative **salience** that ranks DRs as candidates to resolve subsequent definite anaphora

Modelling Components of Contexts

- Following Heim, we model DRs as natural numbers.

Modelling Components of Contexts

- Following Heim, we model DRs as natural numbers.
- Following Stalnaker, we model the CG as a proposition.

Modelling Components of Contexts

- Following Heim, we model DRs as natural numbers.
- Following Stalnaker, we model the CG as a proposition.
- For each $n \in \omega$, we define the type a_n of n-**anchors** as the functions from the first n DRs to entities:

$$\mathrm{a}_n =_{\mathrm{def}} \omega_n \rightarrow \mathrm{e}$$

Modelling Components of Contexts

- Following Heim, we model DRs as natural numbers.
- Following Stalnaker, we model the CG as a proposition.
- For each $n \in \omega$, we define the type a_n of n-**anchors** as the functions from the first n DRs to entities:

$$\mathrm{a}_n =_{\mathrm{def}} \omega_n \rightarrow \mathrm{e}$$

- For each $n \in \omega$, we define the type r_n of n-**resolutions** to be the type of preorders on the first n DRs:

$$\mathrm{r}_n =_{\mathrm{def}} \mathrm{Preord}_{\omega_n}$$

Modelling Contexts

- We model n-contexts (type c_n) as triples of
  - an anchor for the first n DRs
  - a resolution preorder on the first n DRs
  - a proposition (the CG)

Modelling Contexts

- We model n-contexts (type c_n) as triples of
  - an anchor for the first n DRs
  - a resolution preorder on the first n DRs
  - a proposition (the CG)
- Thus:

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{a}_n \times \mathrm{r}_n \times \mathrm{p}$$

$$\mathrm{c} =_{\mathrm{def}} \coprod_{n:\omega} \mathrm{c}_n$$

Modelling Contexts

- We model n-contexts (type c_n) as triples of
  - an anchor for the first n DRs
  - a resolution preorder on the first n DRs
  - a proposition (the CG)
- Thus:

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{a}_n \times \mathrm{r}_n \times \mathrm{p}$$

$$\mathrm{c} =_{\mathrm{def}} \coprod_{n:\omega} \mathrm{c}_n$$

- We abbreviate the three projections of contexts by $\mathbf{a} : \mathrm{c} \to \mathrm{a}$, $\mathbf{r} : \mathrm{c} \to \mathrm{r}$, and $\mathbf{p} : \mathrm{c} \to \mathrm{p}$.

Modelling Contexts

- We model n-contexts (type c_n) as triples of
  - an anchor for the first n DRs
  - a resolution preorder on the first n DRs
  - a proposition (the CG)
- Thus:

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{a}_n \times \mathrm{r}_n \times \mathrm{p}$$

$$\mathrm{c} =_{\mathrm{def}} \coprod_{n:\omega} \mathrm{c}_n$$

- We abbreviate the three projections of contexts by $\mathbf{a} : \mathrm{c} \to \mathrm{a}$, $\mathbf{r} : \mathrm{c} \to \mathrm{r}$, and $\mathbf{p} : \mathrm{c} \to \mathrm{p}$.
- We further abbreviate $(\mathbf{a}\, c\, n)$ to $[n]_c$ (the entity anchored to the DR n in context c)

Functions for Introducing Discourse Referents

For each n:

- the functions $\bullet_n : \mathrm{a}_n \to \mathrm{e} \to \mathrm{a}_{(\mathbf{suc}\, n)}$ (written infix) extend an n-anchor to an $(n+1)$-anchor that maps the 'next' DR to a specified entity:

$$\vdash \forall_{n:\omega} \forall_{a:\mathrm{a}_n} \forall_{x:\mathrm{e}} . (a \bullet_n x)\, n = x$$
$$\vdash \forall_{n:\omega} \forall_{a:\mathrm{a}_n} \forall_{x:\mathrm{e}} \forall_{m:\omega_n} . (a \bullet_n x)\, m = (a\, m)$$

Functions for Introducing Discourse Referents

For each n:

- the functions $\bullet_n : \mathrm{a}_n \to \mathrm{e} \to \mathrm{a}_{(\mathbf{suc}\, n)}$ (written infix) extend an n-anchor to an $(n+1)$-anchor that maps the 'next' DR to a specified entity:

$$\vdash \forall_{n:\omega} \forall_{a:\mathrm{a}_n} \forall_{x:\mathrm{e}} . (a \bullet_n x)\, n = x$$
$$\vdash \forall_{n:\omega} \forall_{a:\mathrm{a}_n} \forall_{x:\mathrm{e}} \forall_{m:\omega_n} . (a \bullet_n x)\, m = (a\, m)$$

- the functions $\star_n : \mathrm{r}_n \to \mathrm{r}_{(\mathbf{suc}\, n)}$ add the next DR n to a resolution so that n is incomparable to all $m < n$:

$$\vdash \forall_n \forall_{r:\mathrm{r}_n} \forall_{m:\omega_{(\mathbf{suc}\, n)}} . ((m\, (\star_n r)\, n) \vee (n\, (\star_n r)\, m)) \to m = n$$

Functions for Introducing Discourse Referents

For each n:

- the functions $\bullet_n : \mathrm{a}_n \to \mathrm{e} \to \mathrm{a}_{(\mathbf{suc}\, n)}$ (written infix) extend an n-anchor to an $(n+1)$-anchor that maps the 'next' DR to a specified entity:

$$\vdash \forall_{n:\omega} \forall_{a:\mathrm{a}_n} \forall_{x:\mathrm{e}} . (a \bullet_n x)\, n = x$$
$$\vdash \forall_{n:\omega} \forall_{a:\mathrm{a}_n} \forall_{x:\mathrm{e}} \forall_{m:\omega_n} . (a \bullet_n x)\, m = (a\, m)$$

- the functions $\star_n : \mathrm{r}_n \to \mathrm{r}_{(\mathbf{suc}\, n)}$ add the next DR n to a resolution so that n is incomparable to all $m < n$:

$$\vdash \forall_n \forall_{r:\mathrm{r}_n} \forall_{m:\omega_{(\mathbf{suc}\, n)}} . ((m\, (\star_n r)\, n) \vee (n\, (\star_n r)\, m)) \to m = n$$

- for any n-context c, the 'next' DR is n:

$$\vdash \forall_n \forall_{c:\mathrm{c}_n} . (\mathbf{next}_n\, c) = n$$

Functions for Updating Contexts

For each n,

- the function $::_n\colon \mathrm{c}_n \to \mathrm{e} \to \mathrm{c}_{(\mathbf{suc}\, n)}$ updates an n-context with a new entity by anchoring the next DR to it:

$$::_n =_{\mathrm{def}} \lambda_{cx}.\, \langle (\mathbf{a}\, c) \bullet_n x, \star_n (\mathbf{r}\, c), (\mathbf{p}\, c) \rangle$$

  This function will be the main ingredient of the dynamic existential quantifier.

Functions for Updating Contexts

For each n,

- the function $::_n\colon \mathrm{c}_n \to \mathrm{e} \to \mathrm{c}_{(\mathbf{suc}\, n)}$ updates an n-context with a new entity by anchoring the next DR to it:

$$::_n =_{\mathrm{def}} \lambda_{cx}.\, \langle (\mathbf{a}\, c) \bullet_n x, \star_n (\mathbf{r}\, c), (\mathbf{p}\, c) \rangle$$

  This function will be the main ingredient of the dynamic existential quantifier.

- the function $+_n : \mathrm{c}_n \to \mathrm{c}_n$ updates an n-context by conjoining a new proposition to its CG:

$$+_n \; =_{\mathrm{def}} \lambda_{cp}.\, \langle (\mathbf{a}\, c), (\mathbf{r}\, c), (\mathbf{p}\, c) \text{ and } p \rangle$$

  This function will be the main ingredient in the **dynamicizer** function that converts static predicates (e.g. verb and noun meanings) into their dynamic counterparts.

Static Properties

- We define the **static property** types as follows:

$$\mathrm{p}_0 =_{\mathrm{def}} \mathrm{p}$$

$$\mathrm{p}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \mathrm{e} \rightarrow \mathrm{p}_n$$

Static Properties

- We define the **static property** types as follows:

$$\mathrm{p}_0 =_{\mathrm{def}} \mathrm{p}$$

$$\mathrm{p}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \mathrm{e} \rightarrow \mathrm{p}_n$$

- Examples:

$$\mathsf{rain}, \mathsf{snow} : \mathrm{p}_0$$
$$\mathsf{donkey}, \mathsf{farmer}, \mathsf{bray} : \mathrm{p}_1$$
$$\mathsf{own}, \mathsf{beat} : \mathrm{p}_2$$

Static Properties

- We define the **static property** types as follows:

$$\mathrm{p}_0 =_{\mathrm{def}} \mathrm{p}$$

$$\mathrm{p}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \mathrm{e} \rightarrow \mathrm{p}_n$$

- Examples:

$$\mathsf{rain}, \mathsf{snow} : \mathrm{p}_0$$
$$\mathsf{donkey}, \mathsf{farmer}, \mathsf{bray} : \mathrm{p}_1$$
$$\mathsf{own}, \mathsf{beat} : \mathrm{p}_2$$

- In particular, 0-ary static properties are just (static) propositions.

Static Properties

- We define the **static property** types as follows:

$$\mathrm{p}_0 =_{\mathrm{def}} \mathrm{p}$$

$$\mathrm{p}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \mathrm{e} \rightarrow \mathrm{p}_n$$

- Examples:

$$\mathsf{rain}, \mathsf{snow} : \mathrm{p}_0$$
$$\mathsf{donkey}, \mathsf{farmer}, \mathsf{bray} : \mathrm{p}_1$$
$$\mathsf{own}, \mathsf{beat} : \mathrm{p}_2$$

- In particular, 0-ary static properties are just (static) propositions.
- What should their dynamic counterparts be?

Context-Dependent Propositions

- Adapting the approach of de Groote 2006, we first define the type k of **context-dependent propositions**, hereafter CDPs, as:

$$\mathrm{k} =_{\mathrm{def}} \mathrm{c} \rightharpoonup \mathrm{p}$$

Context-Dependent Propositions

- Adapting the approach of de Groote 2006, we first define the type k of **context-dependent propositions**, hereafter CDPs, as:

$$\mathrm{k} =_{\mathrm{def}} \mathrm{c} \rightharpoonup \mathrm{p}$$

- This is analogous to de Groote's type $\gamma \rightarrow \mathrm{p}$ of **right contexts**, modulo the replacement of his type γ of **left contexts** with our 'richer' type c of contexts.

Context-Dependent Propositions

- Adapting the approach of de Groote 2006, we first define the type k of **context-dependent propositions**, hereafter CDPs, as:

$$\mathrm{k} =_{\mathrm{def}} \mathrm{c} \rightharpoonup \mathrm{p}$$

- This is analogous to de Groote's type $\gamma \rightarrow \mathrm{p}$ of **right contexts**, modulo the replacement of his type γ of **left contexts** with our 'richer' type c of contexts.
- c is a richer type than γ because a γ is just a finite set of entities.

Updates

- We now define the type u of **updates**, also called **dynamic propositions**, as

$$\mathrm{u} =_{\mathrm{def}} \mathrm{k} \to \mathrm{k} = (\mathrm{c} \rightharpoonup \mathrm{p}) \to \mathrm{c} \rightharpoonup \mathrm{p}$$

Updates

- We now define the type u of **updates**, also called **dynamic propositions**, as

$$\mathrm{u} =_{\mathrm{def}} \mathrm{k} \to \mathrm{k} = (\mathrm{c} \rightharpoonup \mathrm{p}) \to \mathrm{c} \rightharpoonup \mathrm{p}$$

- Modulo replacement of γ by c, and a reversal of the order of arguments, this is analogous to de Groote's type

$$\Omega =_{\mathrm{def}} \gamma \to (\gamma \to \mathrm{p}) \to \mathrm{p}.$$

Updates

- We now define the type u of **updates**, also called **dynamic propositions**, as

$$\mathrm{u} =_{\mathrm{def}} \mathrm{k} \to \mathrm{k} = (\mathrm{c} \rightharpoonup \mathrm{p}) \to \mathrm{c} \rightharpoonup \mathrm{p}$$

- Modulo replacement of γ by c, and a reversal of the order of arguments, this is analogous to de Groote's type

$$\Omega =_{\mathrm{def}} \gamma \to (\gamma \to \mathrm{p}) \to \mathrm{p}.$$

- As with de Groote's right contexts, intuitively the first (CDP) argument should be thought of as corresponding to the continuation of the discourse.

Updates vs. Context Changes

- What is the connection between updates (type $u =_{def} (c \to p) \to (c \to p)$) and the more usual idea of a dynamic sentence meaning as a context change (type $c \to c$)?

Updates vs. Context Changes

- What is the connection between updates (type $\mathrm{u} =_{\mathrm{def}} (\mathrm{c} \to \mathrm{p}) \to (\mathrm{c} \to \mathrm{p})$) and the more usual idea of a dynamic sentence meaning as a context change (type $\mathrm{c} \to \mathrm{c}$)?
- Every context change f uniquely determines an update by the 'contrapositive' embedding $\mu : (\mathrm{c} \to \mathrm{c}) \to \mathrm{u}$ defined as follows:

$$\mu =_{\mathrm{def}} \lambda_{fkc}.(k\ (f\ c))$$

Updates vs. Context Changes

- What is the connection between updates (type $\mathrm{u} =_{\mathrm{def}} (\mathrm{c} \to \mathrm{p}) \to (\mathrm{c} \to \mathrm{p})$) and the more usual idea of a dynamic sentence meaning as a context change (type $\mathrm{c} \to \mathrm{c}$)?
- Every context change f uniquely determines an update by the 'contrapositive' embedding $\mu : (\mathrm{c} \to \mathrm{c}) \to \mathrm{u}$ defined as follows:

$$\mu =_{\mathrm{def}} \lambda_{fkc}.(k\ (f\ c))$$

- That is, for each context change f, the corresponding update $(\mu\ f)$ maps any CDP k to the CDP $k \circ f = \lambda_c.k\ (f\ c)$.

Updates vs. Context Changes

- What is the connection between updates (type $\mathrm{u} =_{\mathrm{def}} (\mathrm{c} \to \mathrm{p}) \to (\mathrm{c} \to \mathrm{p})$) and the more usual idea of a dynamic sentence meaning as a context change (type $\mathrm{c} \to \mathrm{c}$)?
- Every context change f uniquely determines an update by the 'contrapositive' embedding $\mu : (\mathrm{c} \to \mathrm{c}) \to \mathrm{u}$ defined as follows:

$$\mu =_{\mathrm{def}} \lambda_{fkc}.(k\ (f\ c))$$

- That is, for each context change f, the corresponding update $(\mu\ f)$ maps any CDP k to the CDP $k \circ f = \lambda_c.k\ (f\ c)$.
- We ignore context changes and define the updates we use directly.

Dynamic Properties

- Generalizing Muskens 1996, we treat n-ary dynamic properties as functions from n DRs to updates:

$$\mathrm{d}_0 =_{\mathrm{def}} \mathrm{u}$$

$$\mathrm{d}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \omega \rightarrow \mathrm{d}_n$$

Dynamic Properties

- Generalizing Muskens 1996, we treat n-ary dynamic properties as functions from n DRs to updates:

$$\mathrm{d}_0 =_{\mathrm{def}} \mathrm{u}$$

$$\mathrm{d}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \omega \rightarrow \mathrm{d}_n$$

- In particular, 0-ary dynamic properties are just updates.

Dynamic Properties

- Generalizing Muskens 1996, we treat n-ary dynamic properties as functions from n DRs to updates:

$$\mathrm{d}_0 =_{\mathrm{def}} \mathrm{u}$$

$$\mathrm{d}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \omega \rightarrow \mathrm{d}_n$$

- In particular, 0-ary dynamic properties are just updates.
- We abbreviate d_1 (unary dynamic properties) to d.

Dynamic Properties

- Generalizing Muskens 1996, we treat n-ary dynamic properties as functions from n DRs to updates:

$$\mathrm{d}_0 =_{\mathrm{def}} \mathrm{u}$$

$$\mathrm{d}_{(\mathbf{suc}\, n)} =_{\mathrm{def}} \omega \rightarrow \mathrm{d}_n$$

- In particular, 0-ary dynamic properties are just updates.
- We abbreviate d_1 (unary dynamic properties) to d.
- So far we don't actually *have* any dynamic properties, so we will define some functions for converting the static properties we already have into dynamic ones.

Dynamicizing Static Properties

- We recursively define the family of **dynamicizer** functions $\mathbf{dyn}_n : \mathrm{p}_n \to \mathrm{d}_n$ as follows:

$$\mathbf{dyn}_0 =_{\mathrm{def}} \lambda_{pk}.\lambda_{c \mid k\downarrow c+p}.p \text{ and } (k\ (c+p)) : \mathrm{p} \to \mathrm{u}$$

Dynamicizing Static Properties

- We recursively define the family of **dynamicizer** functions $\mathbf{dyn}_n : \mathrm{p}_n \rightarrow \mathrm{d}_n$ as follows:

$$\mathbf{dyn}_0 =_{\mathrm{def}} \lambda_{pk}.\lambda_{c \mid k\downarrow c+p}.p \text{ and } (k\ (c+p)) : \mathrm{p} \rightarrow \mathrm{u}$$

$$\mathbf{dyn}_{(\mathbf{suc}\ n)} =_{\mathrm{def}} \lambda_{Rm}.(\mathbf{dyn}_n\ (R\ [m])) : \mathrm{p}_{(\mathbf{suc}\ n)} \rightarrow \mathrm{d}_{(\mathbf{suc}\ n)}$$

Dynamicizing Static Properties

- We recursively define the family of **dynamicizer** functions $\mathbf{dyn}_n : \mathrm{p}_n \to \mathrm{d}_n$ as follows:

$$\mathbf{dyn}_0 =_{\mathrm{def}} \lambda_{pk}.\lambda_{c \mid k\downarrow c+p}.p \text{ and } (k\ (c+p)) : \mathrm{p} \to \mathrm{u}$$

$$\mathbf{dyn}_{(\mathbf{suc}\ n)} =_{\mathrm{def}} \lambda_{Rm}.(\mathbf{dyn}_n\ (R\ [m])) : \mathrm{p}_{(\mathbf{suc}\ n)} \to \mathrm{d}_{(\mathbf{suc}\ n)}$$

- The restriction on the context variable in the definition of $\mathbf{dyn}_0$ is required to ensure that the body of the abstract always makes sense.
- Context variables like this one which are imposed by the body of the abstract are usually not explicitly written but just understood to be there.

Some Dynamic Properties

- Examples

$$\text{RAIN} =_{\text{def}} (\mathbf{dyn}_0\ \mathsf{rain}) = \lambda_{kc}.\mathsf{rain}\ \mathsf{and}\ (k\ (c + \mathsf{rain}))$$

Some Dynamic Properties

- Examples

$$\text{RAIN} =_{\text{def}} (\mathbf{dyn}_0\ \mathsf{rain}) = \lambda_{kc}.\mathsf{rain}\ \mathsf{and}\ (k\ (c + \mathsf{rain}))$$

$$\text{DONKEY} =_{\text{def}} (\mathbf{dyn}_1\ \mathsf{donkey}) =$$
$$\lambda_{nkc}.(\mathsf{donkey}\ [n])\ \mathsf{and}\ (k\ (c + (\mathsf{donkey}\ [n])))$$

Some Dynamic Properties

- Examples

$$\text{RAIN} =_{\text{def}} (\mathbf{dyn}_0\ \mathsf{rain}) = \lambda_{kc}.\mathsf{rain}\ \mathsf{and}\ (k\ (c + \mathsf{rain}))$$

$$\text{DONKEY} =_{\text{def}} (\mathbf{dyn}_1\ \mathsf{donkey}) =$$
$$\lambda_{nkc}.(\mathsf{donkey}\ [n])\ \mathsf{and}\ (k\ (c + (\mathsf{donkey}\ [n])))$$

$$\text{OWN} =_{\text{def}} (\mathbf{dyn}_2\ \mathsf{own}) =$$
$$\lambda_{mnkc}.(\mathsf{own}\ [m]\ [n])\ \mathsf{and}\ (k\ (c + (\mathsf{own}\ [m]\ [n])))$$

Some Dynamic Properties

- Examples

$$\text{RAIN} =_{\text{def}} (\mathbf{dyn}_0\ \mathsf{rain}) = \lambda_{kc}.\mathsf{rain}\ \mathsf{and}\ (k\ (c + \mathsf{rain}))$$

$$\text{DONKEY} =_{\text{def}} (\mathbf{dyn}_1\ \mathsf{donkey}) = \lambda_{nkc}.(\mathsf{donkey}\ [n])\ \mathsf{and}\ (k\ (c + (\mathsf{donkey}\ [n])))$$

$$\text{OWN} =_{\text{def}} (\mathbf{dyn}_2\ \mathsf{own}) = \lambda_{mnkc}.(\mathsf{own}\ [m]\ [n])\ \mathsf{and}\ (k\ (c + (\mathsf{own}\ [m]\ [n])))$$

- Dynamicization is designed to ensure that asserted propositions get added to the common ground of the discourse continuation.

Dynamic Conjunction

- As usual in this line of work, **dynamic conjunction** is just composition of updates:

$$\text{AND} =_{\text{def}} \lambda_{uvk}.u\,(v\,k)$$

Dynamic Conjunction

- As usual in this line of work, **dynamic conjunction** is just composition of updates:

$$\text{AND} =_{\text{def}} \lambda_{uvk}.u\,(v\,k)$$

- Example: *It rains. It snows.* $\rightsquigarrow$
  $\text{RAIN AND SNOW} = \lambda_k.(\mathbf{dyn}_0\ \mathsf{rain})((\mathbf{dyn}_0\ \mathsf{snow})\ k)$
  $= \lambda_k.(\lambda_{kc}.\mathsf{rain\ and}\ (k\,(c + \mathsf{rain})))(\lambda_c.\mathsf{snow\ and}\ (k\,(c + \mathsf{snow})))$
  $= \lambda_{kc}.\mathsf{rain\ and\ snow\ and}\ (k\,(c + \mathsf{rain} + \mathsf{snow}))$

Dynamic Conjunction

- As usual in this line of work, **dynamic conjunction** is just composition of updates:

$$\text{AND} =_{\text{def}} \lambda_{uvk}.u\,(v\,k)$$

- Example: *It rains. It snows.* $\rightsquigarrow$
  $\text{RAIN AND SNOW} = \lambda_k.(\mathbf{dyn}_0\,\mathsf{rain})((\mathbf{dyn}_0\,\mathsf{snow})\,k)$
  $= \lambda_k.(\lambda_{kc}.\mathsf{rain\ and}\ (k\,(c+\mathsf{rain})))(\lambda_c.\mathsf{snow\ and}\ (k\,(c+\mathsf{snow})))$
  $= \lambda_{kc}.\mathsf{rain\ and\ snow\ and}\ (k\,(c+\mathsf{rain}+\mathsf{snow}))$
- Note that the CG of the context $(c+\mathsf{rain})$ passed to the second conjunct (SNOW) contains the static propositional content (rain) of the first conjunct.

Dynamic Conjunction

- As usual in this line of work, **dynamic conjunction** is just composition of updates:

$$\text{AND} =_{\text{def}} \lambda_{uvk}.u\,(v\,k)$$

- Example: *It rains. It snows.* $\rightsquigarrow$
  $\text{RAIN AND SNOW} = \lambda_k.(\mathbf{dyn}_0\,\mathsf{rain})((\mathbf{dyn}_0\,\mathsf{snow})\,k)$
  $= \lambda_k.(\lambda_{kc}.\mathsf{rain\ and}\ (k\,(c+\mathsf{rain})))(\lambda_c.\mathsf{snow\ and}\ (k\,(c+\mathsf{snow})))$
  $= \lambda_{kc}.\mathsf{rain\ and\ snow\ and}\ (k\,(c+\mathsf{rain}+\mathsf{snow}))$
- Note that the CG of the context $(c+\mathsf{rain})$ passed to the second conjunct (SNOW) contains the static propositional content (rain) of the first conjunct.
- Here nothing hinges on it, but in general this has the consequence that presuppositions of the second conjunct can be satisfied by the first conjunct, e.g. *A donkey$_i$ enters. It$_i$ brays.* but *#It$_i$ brays. A donkey$_i$ enters.*.

Dynamic Existential Quantifier

- The **dynamic existential quantifier** is defined as follows:

$$\text{EXISTS} =_{\text{def}} \lambda_{Dkc}.\mathsf{exists}\,\lambda_x.D\,(\mathbf{next}\,c)\,k\,(c :: x) : \mathrm{d} \rightarrow \mathrm{u}$$

Dynamic Existential Quantifier

- The **dynamic existential quantifier** is defined as follows:

$$\text{EXISTS} =_{\text{def}} \lambda_{Dkc}.\mathsf{exists}\,\lambda_x.D\,(\mathbf{next}\,c)\,k\,(c :: x) : \text{d} \rightarrow \text{u}$$

- Note that what is quantified over is possible anchors for the newly introduced DR.

Dynamic Existential Quantifier

- The **dynamic existential quantifier** is defined as follows:

$$\textsc{exists} =_{\text{def}} \lambda_{Dkc}.\mathsf{exists}\, \lambda_x.D\ (\mathbf{next}\ c)\ k\ (c :: x) : \mathrm{d} \rightarrow \mathrm{u}$$

- Note that what is quantified over is possible anchors for the newly introduced DR.
- The indefinite article *a* is assigned this dynamic meaning, which maps two dynamic properties (restrictor and scope) to an update:

$$\textsc{a} =_{\text{def}} \lambda_{DE}.\textsc{exists}\ \lambda_n.(D\ n)\ \textsc{and}\ (E\ n) : \mathrm{d} \rightarrow \mathrm{d} \rightarrow \mathrm{u}$$

- A is defined in terms of conjunction, which will ensure that a presupposition of the scope can be satisfied in the restrictor e.g. *a donkey denied it brayed.*

Example with an Indefinite

$$a\ donkey\ enters \rightsquigarrow \text{A DONKEY ENTER}$$
$$= \text{EXISTS}\ \lambda_n.(\text{DONKEY}\ n)\ \text{AND}\ (\text{ENTER}\ n)$$
$$= \lambda_{kc}.\mathsf{exists}\ \lambda_x.((\text{DONKEY}\ (\ c))\ \text{AND}\ (\text{ENTER}\ (\ c)))\ k\ (c :: x)$$
$$= \lambda_{kc}.\mathsf{exists}\ \lambda_x.(\mathsf{donkey}\ x)\ \mathsf{and}\ (\mathsf{enter}\ x)$$
$$\mathsf{and}\ (k\ ((c :: x) + \mathsf{donkey}\ x + \mathsf{enter}\ x))$$

Example with an Indefinite

$$
\begin{aligned}
&\textit{a donkey enters} \rightsquigarrow \textsc{a donkey enter} \\
&= \textsc{exists}\ \lambda_n.(\textsc{donkey}\ n)\ \textsc{and}\ (\textsc{enter}\ n) \\
&= \lambda_{kc}.\mathsf{exists}\ \lambda_x.((\textsc{donkey}\ (\ c))\ \textsc{and}\ (\textsc{enter}\ (\ c)))\ k\ (c :: x) \\
&= \lambda_{kc}.\mathsf{exists}\ \lambda_x.(\mathsf{donkey}\ x)\ \mathsf{and}\ (\mathsf{enter}\ x) \\
&\quad \mathsf{and}\ (k\ ((c :: x) + \mathsf{donkey}\ x + \mathsf{enter}\ x))
\end{aligned}
$$

- For each possible choice of anchor c for the new DR (**next** c), the propositions ($\mathsf{donkey}\ x$) and ($\mathsf{enter}\ x$) are included in the context that gets passed to the rest of the discourse.

Example with an Indefinite

$$a\ donkey\ enters \rightsquigarrow \text{A DONKEY ENTER}$$
$$= \text{EXISTS}\ \lambda_n.(\text{DONKEY}\ n)\ \text{AND}\ (\text{ENTER}\ n)$$
$$= \lambda_{kc}.\mathsf{exists}\ \lambda_x.((\text{DONKEY}\ (\ c))\ \text{AND}\ (\text{ENTER}\ (\ c)))\ k\ (c :: x)$$
$$= \lambda_{kc}.\mathsf{exists}\ \lambda_x.(\mathsf{donkey}\ x)\ \mathsf{and}\ (\mathsf{enter}\ x)$$
$$\mathsf{and}\ (k\ ((c :: x) + \mathsf{donkey}\ x + \mathsf{enter}\ x))$$

- For each possible choice of anchor c for the new DR (**next** c), the propositions ($\mathsf{donkey}\ x$) and ($\mathsf{enter}\ x$) are included in the context that gets passed to the rest of the discourse.
- That will enable the new DR to antecede not only a pronoun, but also a definite description such as *the donkey that entered*.

Functions for Handling Presupposition

- the **definedness check** $\downarrow$
- the **staticizer** function **stat**
- the **definiteness** function **def**

The Definedness Check

- The **definedness** function

$$\downarrow =_{\text{def}} \lambda_{kc}.(\mathsf{dom}\ k\ c) : \mathrm{k} \to \mathrm{c} \to \mathrm{t}$$

(written infix) maps a CPD (which is a partial function on contexts) to the characteristic function of its domain.

The Definedness Check

- The **definedness** function

$$\downarrow =_{\text{def}} \lambda_{kc}.(\mathsf{dom}\ k\ c) : \mathrm{k} \to \mathrm{c} \to \mathrm{t}$$

  (written infix) maps a CPD (which is a partial function on contexts) to the characteristic function of its domain.
- Thus $\downarrow$ checks whether a context is in the domain of a CDP.

The Staticizer Function

- The **trivial** CDP

$$\top =_{\text{def}} \lambda_c.\mathsf{true} : \mathrm{k}$$

corresponds intuitively to the end of the discourse.

The Staticizer Function

- The **trivial** CDP

$$\top =_{\text{def}} \lambda_c.\mathsf{true} : \text{k}$$

corresponds intuitively to the end of the discourse.

- The **staticizer** function

$$\mathbf{stat} =_{\text{def}} \lambda_{cu}.(u \top c) : \text{c} \to \text{u} \rightharpoonup \text{p}$$

is used to recover a static proposition from a context and a (suitable) update by 'pretending' the discourse has come to an end.

The Staticizer Function

- The **trivial** CDP

$$\top =_{\text{def}} \lambda_c.\mathsf{true} : \text{k}$$

corresponds intuitively to the end of the discourse.

- The **staticizer** function

$$\mathbf{stat} =_{\text{def}} \lambda_{cu}.(u \top c) : \text{c} \rightarrow \text{u} \rightharpoonup \text{p}$$

is used to recover a static proposition from a context and a (suitable) update by 'pretending' the discourse has come to an end.

- Theorem (easy):

$$\vdash \forall_c \forall_p.(\mathbf{stat}\ c\ (\mathbf{dyn}_0\ p)) \equiv p$$

where $\equiv$ is truth-conditional equivalence (mutual entailment).

The Definiteness Function

- For each n, the **definiteness** function

$$\mathbf{def}_n =_{\text{def}} \lambda_{cD}.\bigsqcup_{(\mathbf{r}\ c)} \lambda_{i:\omega_n}.(\mathbf{p}\ c)\ \mathsf{entails}\ (\mathbf{stat}\ c\ (D\ i)) : \mathrm{c}_n \to \mathrm{d} \to \omega_n$$

  maps an n-context c and a dynamic property D to the most salient DR entailed by c's CG to have that property.

The Definiteness Function

- For each n, the **definiteness** function

$$\mathbf{def}_n =_{\text{def}} \lambda_{cD}.\bigsqcup_{(\mathbf{r}\ c)} \lambda_{i:\omega_n}.(\mathbf{p}\ c)\ \mathsf{entails}\ (\mathbf{stat}\ c\ (D\ i)) : \mathrm{c}_n \to \mathrm{d} \to \omega_n$$

  maps an n-context c and a dynamic property D to the most salient DR entailed by c's CG to have that property.

- This function is called by the dynamic meanings of definite noun phrases, such such as pronouns:

$$\textsc{it} =_{\text{def}} \lambda_{Dkc}.D\ (\mathbf{def}\ c\ \textsc{nonhuman})\ k\ c : \mathrm{d} \to \mathrm{u}$$

  where $\textsc{nonhuman} =_{\text{def}} (\mathbf{dyn}_1\ \mathsf{nonhuman})$

The Definiteness Function

- For each n, the **definiteness** function

$$\mathbf{def}_n =_{\text{def}} \lambda_{cD}.\bigsqcup_{(\mathbf{r}\ c)} \lambda_{i:\omega_n}.(\mathbf{p}\ c)\ \mathsf{entails}\ (\mathbf{stat}\ c\ (D\ i)) : \mathrm{c}_n \to \mathrm{d} \to \omega_n$$

  maps an n-context c and a dynamic property D to the most salient DR entailed by c's CG to have that property.

- This function is called by the dynamic meanings of definite noun phrases, such such as pronouns:

$$\textsc{it} =_{\text{def}} \lambda_{Dkc}.D\ (\mathbf{def}\ c\ \textsc{nonhuman})\ k\ c : \mathrm{d} \to \mathrm{u}$$

  where $\textsc{nonhuman} =_{\text{def}} (\mathbf{dyn}_1\ \mathsf{nonhuman})$

- *It brays.* $\rightsquigarrow (\textsc{it bray}) =$

$$\lambda_{kc}.\mathsf{bray}[\mathbf{def}\ c\ \textsc{nonhuman}]\ \mathsf{and}\ (k\ (c + \mathsf{bray}[\mathbf{def}\ c\ \textsc{nonhuman}]))$$

A Definite Anaphora Example

- *A donkey enters. It brays.* $\rightsquigarrow$

A Definite Anaphora Example

- *A donkey enters. It brays.* $\rightsquigarrow$

$$(\text{A DONKEY ENTER}) \text{ AND } (\text{IT BRAY}) =$$
$$\lambda_{kc}.\mathsf{exists}\ \lambda_x.(\mathsf{donkey}\ x)\ \mathsf{and}\ (\mathsf{enter}\ x)$$
$$\mathsf{and}\ \mathsf{bray}[\mathbf{def}\ c'[c,x]\ \text{NONUMAN}]$$
$$\mathsf{and}\ (k\ (c'[c,x] + \mathsf{bray}[\mathbf{def}\ c'[c,x]\ \ \text{NONHUMAN}]))$$

where $c'[c,x]$ is $(c :: x) + \mathsf{donkey}\ x + \mathsf{enter}\ x$.

A Definite Anaphora Example

- *A donkey enters. It brays.* $\rightsquigarrow$

$$(\text{A DONKEY ENTER}) \text{ AND } (\text{IT BRAY}) =$$
$$\lambda_{kc}.\mathsf{exists}\ \lambda_x.(\mathsf{donkey}\ x)\ \mathsf{and}\ (\mathsf{enter}\ x)$$
$$\mathsf{and}\ \mathsf{bray}[\mathbf{def}\ c'[c,x]\ \text{NONUMAN}]$$
$$\mathsf{and}\ (k\ (c'[c,x] + \mathsf{bray}[\mathbf{def}\ c'[c,x]\ \text{ NONHUMAN}]))$$

  where $c'[c,x]$ is $(c :: x) + \mathsf{donkey}\ x + \mathsf{enter}\ x$.
- As long as (1) it is in the CG that donkeys are nonhuman, and (2) no inferrably nonhuman DR more salient than (**next** c) is present, then $[\mathbf{def}\ c'[c,x]\ \text{NONHUMAN}] = x$.
- Details are in Martin and Pollard 2010 (Formal Grammar paper).

Presupposition Projection

- It's well known that many constructions inherit the presuppositions of an embedded expression.

Presupposition Projection

- It's well known that many constructions inherit the presuppositions of an embedded expression.
- E.g. presuppositions 'project' through negation:
  1. A donkey entered. It brayed.
  2. A donkey entered. It didn't bray.
  3. #It brayed. (out of the blue)
  4. #It didn't bray. (out of the blue)

Presupposition Projection

- It's well known that many constructions inherit the presuppositions of an embedded expression.
- E.g. presuppositions 'project' through negation:
  1. A donkey entered. It brayed.
  2. A donkey entered. It didn't bray.
  3. #It brayed. (out of the blue)
  4. #It didn't bray. (out of the blue)
  5. Kim is going to the party. Sandy is going too.
  5. Kim is going to the party. No way Sandy is going too.
  7. #Sandy is going too. (out of the blue)
  8. #No way Sandy is going too. (out of the blue)

Dynamic Negation

- Analogizing directly from de Groote would give the following dynamic meaning:

$$\text{NOT} =_{\text{def}} \lambda_{ukc}.\mathbf{dyn}_0(\mathsf{not}\,(\mathbf{stat}\,c\,u))\,k\,c : \text{u} \rightarrow \text{u}$$

Dynamic Negation

- Analogizing directly from de Groote would give the following dynamic meaning:

$$\text{NOT} =_{\text{def}} \lambda_{ukc}.\mathbf{dyn}_0(\mathsf{not}\,(\mathbf{stat}\,c\,u))\,k\,c : \text{u} \rightarrow \text{u}$$

- We amend this to:

$$\lambda_{uk}.\lambda_{c \mid (u\,k)\downarrow c}.\mathbf{dyn}_0(\mathsf{not}\,(\mathbf{stat}\,c\,u))\,k\,c$$

- The restriction on the context variable requires that the denial be defined in the same contexts that the update being denied is defined.

Factivity

- Compare these single-speaker discourses:

    1. Pedro thinks it's raining. But it's not raining.
    2. It sucks that it's raining. #But it's not raining.

Factivity

- Compare these single-speaker discourses:

    1. Pedro thinks it's raining. But it's not raining.
    2. It sucks that it's raining. #But it's not raining.

- The difference is not simply that *it sucks that it's raining entails* that it's raining, since the implication projects through negation:

    3. It doesn't suck that it's raining. #But it's not raining.

Factivity

- Compare these single-speaker discourses:

    1. Pedro thinks it's raining. But it's not raining.
    2. It sucks that it's raining. #But it's not raining.

- The difference is not simply that *it sucks that it's raining* *entails* that it's raining, since the implication projects through negation:

    3. It doesn't suck that it's raining. #But it's not raining.

- The difference is that *sucks* is **factive** (presupposes the proposition expressed by its sentential complement), so that the second assertion contradicts the CG.

Analysis of Factivity

- A naive dynamic meaning for *suck* would be:

$$\text{SUCK} =_{\text{def}} \lambda_{ukc}.\mathbf{dyn}_0(\mathsf{suck}\,(\mathbf{stat}\,c\,u))\,k\,c : \text{u} \rightarrow \text{u}$$

Analysis of Factivity

- A naive dynamic meaning for *suck* would be:

$$\text{SUCK} =_{\text{def}} \lambda_{ukc}.\mathbf{dyn}_0(\mathsf{suck}\ (\mathbf{stat}\ c\ u))\ k\ c : \text{u} \rightarrow \text{u}$$

- We amend this to:

$$\lambda_{uk}.\lambda_{c\ |\ (\mathbf{p}\ c)\ \mathsf{entails}\ (\mathbf{stat}\ c\ u)}.\mathbf{dyn}_0(\mathsf{suck}\ (\mathbf{stat}\ c\ u))\ k\ c$$

Analysis of Factivity

- A naive dynamic meaning for *suck* would be:

$$\text{SUCK} =_{\text{def}} \lambda_{ukc}.\mathbf{dyn}_0(\mathsf{suck}\ (\mathbf{stat}\ c\ u))\ k\ c : \text{u} \rightarrow \text{u}$$

- We amend this to:

$$\lambda_{uk}.\lambda_{c\ |\ (\mathbf{p}\ c)\ \mathsf{entails}\ (\mathbf{stat}\ c\ u)}.\mathbf{dyn}_0(\mathsf{suck}\ (\mathbf{stat}\ c\ u))\ k\ c$$

- The restriction on the context variable requires that its CG entail the proposition expressed by the sentential complement.

Presuppositions of Conditionals

- In a conditional sentence, the antecedent can satisfy the presuppositions of the consequent:
    1. If a donkey enters, it brays.
    2. If it's raining, that (it's raining) sucks.
    3. If Kim is going to the party, Sandy is going too.

Presuppositions of Conditionals

- In a conditional sentence, the antecedent can satisfy the presuppositions of the consequent:
    1. If a donkey enters, it brays.
    2. If it's raining, that (it's raining) sucks.
    3. If Kim is going to the party, Sandy is going too.
- In a conditional sentence, it is presupposed that neither the antecedent not its denial is entailed by the CG.

Presuppositions of Conditionals

- In a conditional sentence, the antecedent can satisfy the presuppositions of the consequent:
  1. If a donkey enters, it brays.
  2. If it's raining, that (it's raining) sucks.
  3. If Kim is going to the party, Sandy is going too.
- In a conditional sentence, it is presupposed that neither the antecedent not its denial is entailed by the CG.
  Context: The speaker and the addressee are watching a huge rainstorm out the window.
  1. #If it's raining, my convertible is getting ruined.
  2. #If it's not raining, my convertible is getting ruined.

Analysis of Presuppositions of Conditionals (1/2)

- We start with de Groote's dynamic conditional semantics:

$$\text{IF} =_{\text{def}} \lambda_{uv}.\text{NOT}\,(u\ \text{AND}\ (\text{NOT}\ v)) : \text{u} \rightarrow \text{u} \rightarrow \text{u}$$

Analysis of Presuppositions of Conditionals (1/2)

- We start with de Groote's dynamic conditional semantics:

$$\text{IF} =_{\text{def}} \lambda_{uv}.\text{NOT}\,(u\ \text{AND}\ (\text{NOT}\ v)) : \text{u} \rightarrow \text{u} \rightarrow \text{u}$$

- Theorem: In the simple case where u and v are the dynamicizations of propositions p and q respectively (and therefore have no presuppositions), we have:

$$\vdash \text{IF}\ u\ v = \lambda_{kc}.\mathsf{not}\,(p\ \mathsf{and}\ (\mathsf{not}\ q)) \equiv \lambda_{kc}.p\ \mathsf{implies}\ q$$

Analysis of Presuppositions of Conditionals (1/2)

- We start with de Groote's dynamic conditional semantics:

$$\text{IF} =_{\text{def}} \lambda_{uv}.\text{NOT}\,(u\ \text{AND}\ (\text{NOT}\ v)) : \text{u} \rightarrow \text{u} \rightarrow \text{u}$$

- Theorem: In the simple case where u and v are the dynamicizations of propositions p and q respectively (and therefore have no presuppositions), we have:

$$\vdash \text{IF}\ u\ v = \lambda_{kc}.\mathsf{not}\ (p\ \mathsf{and}\ (\mathsf{not}\ q)) \equiv \lambda_{kc}.p\ \mathsf{implies}\ q$$

- We *already* predict that the antecedent can satisfy presuppositions of the consequent, since we know that
  - the first conjunct of a conjunction can satisfy the presuppositions of the second conjunct, and
  - presuppositions project through negation.

Analysis of Presuppositions of Conditionals (1/2)

- We start with de Groote's dynamic conditional semantics:

$$\text{IF} =_{\text{def}} \lambda_{uv}.\text{NOT}\,(u\ \text{AND}\ (\text{NOT}\ v)) : \text{u} \rightarrow \text{u} \rightarrow \text{u}$$

- Theorem: In the simple case where u and v are the dynamicizations of propositions p and q respectively (and therefore have no presuppositions), we have:

$$\vdash \text{IF}\ u\ v = \lambda_{kc}.\mathsf{not}\,(p\ \mathsf{and}\ (\mathsf{not}\ q)) \equiv \lambda_{kc}.p\ \mathsf{implies}\ q$$

- We *already* predict that the antecedent can satisfy presuppositions of the consequent, since we know that
  - the first conjunct of a conjunction can satisfy the presuppositions of the second conjunct, and
  - presuppositions project through negation.
- However, this does not address the issue of the independence of the antecedent from the CG.

Analysis of Presuppositions of Conditionals (2/2)

- We handle this by amending the definition of IF to be:

$$\lambda_{uvk}.\lambda_{c \mid (\mathbf{stat}\, c\, u)\ \mathsf{indep}\ (\mathbf{p}\, c)}.\text{NOT}\ (u\ \text{AND}\ (\text{NOT}\ v))\ k\ c$$

  where $\mathsf{indep} : \mathrm{p} \to \mathrm{p} \to \mathrm{t}$ is defined as:

$$\lambda_{pq}.\neg((p\ \mathsf{entails}\ q) \vee (p\ \mathsf{entails}\ (\mathsf{not}\ q)))$$

- The condition on the context variable has as a consequence that neither the antecedent of the conditional nor its denial is entailed by the common ground.

To Do Next

You name it, we've barely scratched the surface.