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

Hyperintensional Dynamic Semantics
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
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

- Handles phenomena not handled by ‘static’ semantics (no matter whether Montagovian or hyperintensional):
  - novelty condition on indefinites
  - cross-sentential anaphora
  - donkey anaphora
Sentence meanings are relations (or partial functions) between contexts.

Sentence conjunction is interpreted as composition.
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.
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.
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.
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
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.
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).
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
We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types $t$ (truth values) and $\omega$ (natural numbers)
- the usual cartesian-closed type constructors $1$ (unit), $\times$ (product), and $\to$ (exponential)
We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types \( t \) (truth values) and \( \omega \) (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 \( \{x \in A \mid \varphi[x]\} \).
We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types $t$ (truth values) and $\omega$ (natural numbers)
- the usual cartesian-closed type constructors $1$ (unit), $\times$ (product), and $\to$ (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 $I(\lambda_x.\varphi[x])$. 

Carl Pollard
Hyperintensional Dynamic Semantics
We work in a classical HOL along the lines of Lambek and Scott (1986):

- basic types $t$ (truth values) and $\omega$ (natural numbers)
- the usual cartesian-closed type constructors $1$ (unit), $\times$ (product), and $\to$ (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 $\omega$), written $\coprod_{n:\omega} A_n$. 
\( \omega_n = \text{def} \{ i \in \omega \mid i < n \} \): the first \( n \) natural numbers
Technical Preliminaries: Some Defined Types

- \( \omega_n = \text{def } \{ i \in \omega \mid i < n \} : \text{the first } n \text{ natural numbers} \)
- \( \text{Rel}_{A,B} = \text{def } A \rightarrow B \rightarrow t : \text{relations from } A \text{ to } B \)
Technical Preliminaries: Some Defined Types

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

- \( \omega_n = \{ i \in \omega \mid i < n \} \): the first \( n \) natural numbers
- \( \text{Rel}_{A,B} = \text{def} \ A \rightarrow B \rightarrow t \): relations from \( A \) to \( B \)
- \( A \rightharpoonup B \) (subtype of \( \text{Rel}_{A,B} \)): partial functions from \( A \) to \( B \)
- \( \text{Preord}_A \) (subtype of \( \text{Rel}_{A,A} \)): preorders on \( A \)
Technical Preliminaries: Some Defined Types

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

But entailment is not assumed to be antisymmetric.

The type w of worlds is defined as a subtype of p → t (the ultrafilters).
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.

Carl Pollard  
Hyperintensional Dynamic Semantics
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 → t (the ultrafilters).
The **hyperintensional** types are p, e, and types obtained from these using the type constructors.
The hyperintensional types are p, e, and types obtained from these using the type constructors.

For each hyperintensional type $A$, the corresponding extensional type $\text{Ext}(A)$ is defined as follows:

- $\text{Ext}(p) = t$

Meanings are hyperintensions, not intensions.
The hyperintensional types are p, e, and types obtained from these using the type constructors.

For each hyperintensional type $A$, the corresponding extensional type $\text{Ext}(A)$ is defined as follows:

- $\text{Ext}(p) = t$
- $\text{Ext}(e) = e$

So there are intensions. But meanings are hyperintensions, not intensions.
The **hyperintensional** types are p, e, and types obtained from these using the type constructors.

For each hyperintensional type $A$, the corresponding **extensional** type $\text{Ext}(A)$ is defined as follows:

- $\text{Ext}(p) = t$
- $\text{Ext}(e) = e$
- $\text{Ext}(B \rightarrow C) = B \rightarrow \text{Ext}(C)$

So there are intensions. But meanings are hyperintensions, not intensions.
The hyperintensional types are p, e, and types obtained from these using the type constructors.

For each hyperintensional type $A$, the corresponding extensional type $\text{Ext}(A)$ is defined as follows:

- $\text{Ext}(p) = t$
- $\text{Ext}(e) = e$
- $\text{Ext}(B \to C) = B \to \text{Ext}(C)$

and the corresponding intensional type $\text{Int}(A)$ is defined as $w \to \text{Ext}(A)$. 
The hyperintensional types are \( p, e, \) and types obtained from these using the type constructors.

For each hyperintensional type \( A \), the corresponding extensional type \( \text{Ext}(A) \) is defined as follows:

- \( \text{Ext}(p) = t \)
- \( \text{Ext}(e) = e \)
- \( \text{Ext}(B \rightarrow C) = B \rightarrow \text{Ext}(C) \)

and the corresponding intensional type \( \text{Int}(A) \) is defined as \( w \rightarrow \text{Ext}(A) \).

So there are intensions. But meanings are hyperintensions, not intensions.
If \( a : A \) is a hyperintension and \( w \) a world, the \textbf{extension} of \( a \) at \( w \), written \( a@w : \text{Ext}(A) \), is defined to be:

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

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

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

- \((w a) : t \) if \( A = p \)
- \( a : e \) if \( A = e \)
- \( \lambda_x.(a x)@w : B \to \text{Ext}(C) \) if \( A = B \to C \)

For each hyperintensional type \( A \), the intensionalizer function is \( \text{int}_A = \text{def} \lambda_{wx}.x@w : A \to \text{Int}(A) \) and for each \( a : A \), \( (\text{int} a) \) is called the intension corresponding to \( a \).
If $a : A$ is a hyperintension and $w$ a world, the extension of $a$ at $w$, written $a@w : \text{Ext}(A)$, is defined to be:

- $(w a) : t$ if $A = p$
- $a : e$ if $A = e$
- $\lambda x.(a x)@w : B \to \text{Ext}(C)$ if $A = B \to C$

For each hyperintensional type $A$, the intensionalizer function is $\text{int}_A = \text{def} \lambda wx.x@w : A \to \text{Int}(A)$ and for each $a : A$, $(\text{int} a)$ is called the intension corresponding to $a$.

$\text{int}_p : p \to w \to t$ is the Stone dual mapping that maps each proposition to the set of worlds which contain it.
If $a : A$ is a hyperintension and $w$ a world, the extension of $a$ at $w$, written $a@w : \text{Ext}(A)$, is defined to be:

- $(w \ a) : t$ if $A = p$
- $a : e$ if $A = e$
- $\lambda x. (a \ x)@w : B \rightarrow \text{Ext}(C)$ if $A = B \rightarrow C$

For each hyperintensional type $A$, the intensionalizer function is $\text{int}_A = \text{def} \ \lambda w x. x@w : A \rightarrow \text{Int}(A)$ and for each $a : A$, $(\text{int} \ a)$ is called the intension corresponding to $a$.

$\text{int}_p : p \rightarrow w \rightarrow t$ is the Stone dual mapping that maps each proposition to the set of worlds which contain it.

Since entailment is not antisymmetric, $\text{int}_p$ is many-to-one.
If $a : A$ is a hyperintension and $w$ a world, the extension of $a$ at $w$, written $a@w : \text{Ext}(A)$, is defined to be:

- $(w \thinspace a) : t$ if $A = p$
- $a : e$ if $A = e$
- $\lambda_x.(a \thinspace x)@w : B \rightarrow \text{Ext}(C)$ if $A = B \rightarrow C$

For each hyperintensional type $A$, the intensionalizer function is $\text{int}_A = \text{def} \thinspace \lambda w.x@w : A \rightarrow \text{Int}(A)$ and for each $a : A$, $(\text{int} \thinspace a)$ is called the intension corresponding to $a$.

$\text{int}_p : p \rightarrow w \rightarrow t$ is the Stone dual mapping that maps each proposition to the set of worlds which contain it.

Since entailment is not antisymmetric, $\text{int}_p$ is many-to-one.

That’s why hyperintensional semantics is more fine-grained than Montague semantics.
Hyperintensional Quantifiers

- exists : (e → p) → p

Axiom: \( \vdash \forall P : e \rightarrow p \forall w : w. (\text{exists } P)@w = \exists x . (P x)@w \)
Hyperintensional Quantifiers

- **exists**: $(e \rightarrow p) \rightarrow p$
  Axiom: $\vdash \forall P: e \rightarrow p \forall w:w.(\text{exists } P)@w = \exists_x.(P x)@w$

- **forall**: $(e \rightarrow p) \rightarrow p$
  Axiom: $\vdash \forall P: e \rightarrow p \forall w:w.(\text{forall } P)@w = \forall_x.(P x)@w$
Some Word Translations

*Pedro* ~* pedro : e

Carl Pollard
Hyperintensional Dynamic Semantics
Some Word Translations

Pedro $\sim\sim$ pedro : e

donkey $\sim\sim$ donkey : e $\rightarrow$ p
Some Word Translations

*Pedro* $\leadsto$ *pedro* : $e$

*donkey* $\leadsto$ *donkey* : $e \rightarrow p$

*bray* $\leadsto$ *bray* : $e \rightarrow p$
Some Word Translations

Pedro $\leadsto$ pedro : e

donkey $\leadsto$ donkey : e $\rightarrow$ p

bray $\leadsto$ bray : e $\rightarrow$ p

own $\leadsto$ own : e $\rightarrow$ e $\rightarrow$ p

Carl Pollard

Hyperintensional Dynamic Semantics
Some Word Translations

\[\begin{align*}
\text{Pedro} &\rightsquigarrow \text{pedro} : e \\
\text{donkey} &\rightsquigarrow \text{donkey} : e \rightarrow p \\
\text{bray} &\rightsquigarrow \text{bray} : e \rightarrow p \\
\text{own} &\rightsquigarrow \text{own} : e \rightarrow e \rightarrow p \\
\text{it}_{\text{dummy}} &\rightsquigarrow * : 1
\end{align*}\]
Some Word Translations

Pedro $\leadsto$ pedro : e

donkey $\leadsto$ donkey : e $\rightarrow$ p

bray $\leadsto$ bray : e $\rightarrow$ p

own $\leadsto$ own : e $\rightarrow$ e $\rightarrow$ p

$it_{\text{dummy}}$ $\leadsto$ $\ast : 1$

$\lambda u.\text{rain} : 1 \rightarrow p$ where $\text{rain} : p$
Some Word Translations

Pedro ⇾ pedro : e

*donkey* ⇾ donkey : e → p

*bray* ⇾ bray : e → p

*own* ⇾ own : e → e → p

*it\textsubscript{dummy}* ⇾ * : 1

*rain* ⇾ \( \lambda_u.\text{rain} \) : 1 → p where rain : p

*suck* ⇾ \( \lambda_{pu}.\text{suck} \) : p → 1 → p where suck : p → p
Some Word Translations

Pedro $\leadsto$ pedro : e

*donkey* $\leadsto$ donkey : e $\rightarrow$ p

*bray* $\leadsto$ bray : e $\rightarrow$ p

*own* $\leadsto$ own : e $\rightarrow$ e $\rightarrow$ p

*it*$_{dummy}$ $\leadsto$ * : 1

*rain* $\leadsto$ $\lambda_u$.rain : 1 $\rightarrow$ p where rain : p

*suck* $\leadsto$ $\lambda_{pu}$.suck p : p $\rightarrow$ 1 $\rightarrow$ p where suck : p $\rightarrow$ p

*no way* $\leadsto$ not : p $\rightarrow$ p
Some Word Translations

Pedro $\leadsto$ pedro : e

donkey $\leadsto$ donkey : e $\rightarrow$ p

bray $\leadsto$ bray : e $\rightarrow$ p

own $\leadsto$ own : e $\rightarrow$ e $\rightarrow$ p

it dummy $\leadsto$ * : 1

rain $\leadsto$ $\lambda$u.rain : 1 $\rightarrow$ p where rain : p

suck $\leadsto$ $\lambda$pu.suck p : p $\rightarrow$ 1 $\rightarrow$ p where suck : p $\rightarrow$ p

no way $\leadsto$ not : p $\rightarrow$ p

a $\leadsto$ $\lambda$P,Q.(∃ $\lambda$x.(P x) and (Q x)) : (e $\rightarrow$ p) $\rightarrow$ (e $\rightarrow$ p) $\rightarrow$ p
Some Sentence Translations (Given by the Grammar)

\[
\text{Chiquita brays} \leadsto \text{bray chiquita}
\]
Some Sentence Translations (Given by the Grammar)

Chiquita brays $\leadsto$ bray chiquita

No way Chiquita brays $\leadsto$ not (bray chiquita)
Some Sentence Translations (Given by the Grammar)

\[
\begin{align*}
\text{Chiquita brays} & \rightsquigarrow \text{bray chiquita} \\
\text{No way Chiquita brays} & \rightsquigarrow \text{not (bray chiquita)} \\
\text{Pedro owns Chiquita} & \rightsquigarrow \text{own chiquita pedro}
\end{align*}
\]
Some Sentence Translations (Given by the Grammar)

*Chiquita brays* $\mapsto$ *bray chiquita*

*No way Chiquita brays* $\mapsto$ *not (bray chiquita)*

*Pedro owns Chiquita* $\mapsto$ *own chiquita pedro*

*It rains* $\mapsto$ $(\lambda_u.\text{rain})^* = \text{rain}$
Some Sentence Translations (Given by the Grammar)

\[ Chiquita \text{ brays} \sim \text{bray chiquita} \]
\[ No \text{ way } Chiquita \text{ brays} \sim \text{not (bray chiquita)} \]
\[ Pedro \text{ owns } Chiquita \sim \text{own chiquita pedro} \]
\[ It \text{ rains} \sim (\lambda u. \text{rain}) \ast = \text{rain} \]
\[ It \text{ sucks that it rains} \sim \text{suck rain} \]
Some Sentence Translations (Given by the Grammar)

Chiquita brays $\leadsto$ bray chiquita

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

Pedro owns Chiquita $\leadsto$ own chiquita pedro

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

It sucks that it rains $\leadsto$ suck rain

Pedro owns a donkey $\leadsto$ exists $\lambda_x. (\text{donkey } x)$ and (own $x$ pedro)
Some Sentence Translations (Given by the Grammar)

*Chiquita brays* \(\leadsto\) bray chiquita

No way Chiquita brays \(\leadsto\) not (bray chiquita)

Pedro owns Chiquita \(\leadsto\) own chiquita pedro

It rains \(\leadsto\) (\(\lambda u.\) rain) \(*=\) rain

It sucks that it rains \(\leadsto\) suck rain

Pedro owns a donkey \(\leadsto\) exists \(\lambda x.\) (donkey \(x\)) and (own \(x\) 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)

\[ Chiquita \text{ brays} \implies \text{bray chiquita} \]
\[ \text{No way Chiquita brays} \implies \text{not (bray chiquita)} \]
\[ \text{Pedro owns Chiquita} \implies \text{own chiquita pedro} \]
\[ \text{It rains} \implies (\lambda u. \text{rain}) \ast = \text{rain} \]
\[ \text{It sucks that it rains} \implies \text{suck rain} \]
\[ \text{Pedro owns a donkey} \implies \text{exists } \lambda x. (\text{donkey } x) \text{ and (own } x \text{ pedro)} \]

- 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 $\leadsto$ bray chiquita
No way Chiquita brays $\leadsto$ not (bray chiquita)
Pedro owns Chiquita $\leadsto$ own chiquita pedro
It rains $\leadsto$ ($\lambda u.\text{rain}$) $\ast =$ rain
It sucks that it rains $\leadsto$ suck rain
Pedro owns a donkey $\leadsto$ exists $\lambda x.(\text{donkey } x)$ and (own $x$ pedro)

- Remember that these meanings are propositions (type p), not truth values (type t) or Carnap/Montague-style intensions (type $w \to t$).
- 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.
Following Heim, we model DRs as natural numbers.
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:

\[
a_n = \text{def} \omega_n \rightarrow e
\]
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:

\[
a_n = \text{def } \omega_n \rightarrow e
\]

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:

\[
r_n = \text{def } \text{Preord}_{\omega_n}
\]
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)
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:

$$c_n = \text{def} \ a_n \times r_n \times p$$

$$c = \text{def} \ \bigsqcup_{n: \omega} c_n$$
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:

$$c_n = \text{def } a_n \times r_n \times p$$

$$c = \text{def } \bigsqcup_{n:\omega} c_n$$

We abbreviate the three projections of contexts by

$a : c \rightarrow a$, $r : c \rightarrow r$, and $p : c \rightarrow p$. 
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:

$$c_n = \text{def } a_n \times r_n \times p$$

$$c = \text{def } \bigsqcup_{n: \omega} c_n$$

- We abbreviate the three projections of contexts by $a : c \rightarrow a$, $r : c \rightarrow r$, and $p : c \rightarrow p$.

- We further abbreviate $(a c n)$ to $[n]_c$ (the entity anchored to the DR $n$ in context $c$)
For each \( n \):

- the functions \( \bullet_n : a_n \rightarrow e \rightarrow a_{\text{succ } n} \) (written infix) extend an \( n \)-anchor to an \( (n + 1) \)-anchor that maps the ‘next’ DR to a specified entity:

\[
\vdash \forall n: \omega \forall a:a_n \forall x:e. (a \bullet_n x) \ n = x \\
\vdash \forall n: \omega \forall a:a_n \forall x:e \forall m: \omega_n. (a \bullet_n x) \ m = (a \ m)
\]
For each $n$:

- the functions $\bullet_n : a_n \to e \to a_{(\text{suc } n)}$ (written infix) extend an $n$-anchor to an $(n + 1)$-anchor that maps the ‘next’ DR to a specified entity:

  \[
  \vdash \forall n: \omega \forall a : a_n \forall x : e. (a \bullet_n x) n = x \\
  \vdash \forall n: \omega \forall a : a_n \forall x : e \forall m : \omega_n. (a \bullet_n x) m = (a m)
  \]

- the functions $\star_n : r_n \to r_{(\text{suc } n)}$ add the next DR $n$ to a resolution so that $n$ is incomparable to all $m < n$:

  \[
  \vdash \forall n \forall r : r_n \forall m : \omega_{\text{suc } n}. ((m (\star_n r) n) \lor (n (\star_n r) m)) \to m = n
  \]
For each $n$:

- the functions $\bullet_n : a_n \rightarrow e \rightarrow a_{\text{suc } n}$ (written infix) extend an $n$-anchor to an $(n + 1)$-anchor that maps the ‘next’ DR to a specified entity:

\[
\vdash \forall n : \omega \forall a : a_n \forall x : e. (a \bullet_n x) n = x
\]

\[
\vdash \forall n : \omega \forall a : a_n \forall x : e \forall m : \omega_n. (a \bullet_n x) m = (a m)
\]

- the functions $\star_n : r_n \rightarrow r_{\text{suc } n}$ add the next DR $n$ to a resolution so that $n$ is incomparable to all $m < n$:

\[
\vdash \forall n \forall r : r_n \forall m : \omega_{\text{suc } n}. ((m (\star_n r) n) \lor (n (\star_n r) m)) \rightarrow m = n
\]

- for any $n$-context $c$, the ‘next’ DR is $n$:

\[
\vdash \forall n \forall c : c_n. (\text{next}_n c) = n
\]
Functions for Updating Contexts

For each $n$,

- the function $::_{n} : c_{n} \rightarrow e \rightarrow c_{\text{suc } n}$ updates an $n$-context with a new entity by anchoring the next DR to it:

  $$::_{n} = \text{def } \lambda_{cx}. \langle (a \ c) \bullet_{n} x, \star_{n} (r \ c), (p \ c) \rangle$$

  This function will be the main ingredient of the dynamic existential quantifier.
Functions for Updating Contexts

For each $n$,

- the function $:\!::_n : c_n \to e \to c_{\text{suc} \, n}$ updates an $n$-context with a new entity by anchoring the next DR to it:

  $$\::_n = \text{def} \lambda cx. \langle (a \, c) \bullet n \, x, *_n (r \, c), (p \, c) \rangle$$

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

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

  $$+_n = \text{def} \lambda cp. \langle (a \, c), (r \, c), (p \, c) \, \text{and} \, p \rangle$$

  This function will be the main ingredient in the \textbf{dynamicizer} function that converts static predicates (e.g. verb and noun meanings) into their dynamic counterparts.
We define the static property types as follows:

\[
P_0 \equiv \text{def } P \\
P(n \text{ suc }) \equiv \text{def } e \rightarrow P_n
\]
We define the static property types as follows:

\[ p_0 = \text{def } P \]

\[ P(\text{suc } n) = \text{def } e \rightarrow P_n \]

Examples:

\[ \text{rain, snow : } p_0 \]
\[ \text{donkey, farmer, bray : } p_1 \]
\[ \text{own, beat : } p_2 \]
Static Properties

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

  \[ P_0 = \text{def } P \]
  \[ P(\text{suc } n) = \text{def } e \rightarrow P_n \]

- Examples:

  - rain, snow : \( P_0 \)
  - donkey, farmer, bray : \( P_1 \)
  - own, beat : \( P_2 \)

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

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

\[
P_0 = \text{def } P \\
P\left(\text{suc } n\right) = \text{def } e \rightarrow P_n
\]

- Examples:

  \[
  \text{rain, snow} : p_0 \\
  \text{donkey, farmer, bray} : p_1 \\
  \text{own, beat} : p_2
  \]

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

- What should their dynamic counterparts be?
Adapting the approach of de Groote 2006, we first define the type $k$ of context-dependent propositions, hereafter CDPs, as:

$$k \overset{\text{def}}{=} c \rightarrow p$$
Adapting the approach of de Groote 2006, we first define the type $k$ of context-dependent propositions, hereafter CDPs, as:

$$k = \text{def } c \rightarrow p$$

This is analogous to de Groote’s type $\gamma \rightarrow p$ of right contexts, modulo the replacement of his type $\gamma$ of left contexts with our ‘richer’ type $c$ of contexts.
Adapting the approach of de Groote 2006, we first define the type $k$ of context-dependent propositions, hereafter CDPs, as:

$$k = \text{def} \ c \rightarrow p$$

This is analogous to de Groote’s type $\gamma \rightarrow p$ of right contexts, modulo the replacement of his type $\gamma$ of left contexts with our ‘richer’ type $c$ of contexts.

$c$ is a richer type than $\gamma$ because a $\gamma$ is just a finite set of entities.
We now define the type $u$ of **updates**, also called **dynamic propositions**, as

$$u = \text{def } k \rightarrow k = (c \rightarrow p) \rightarrow c \rightarrow p$$
We now define the type $u$ of **updates**, also called **dynamic propositions**, as

$$u = \text{def } k \rightarrow k = (c \rightarrow p) \rightarrow c \rightarrow p$$

Modulo replacement of $\gamma$ by $c$, and a reversal of the order of arguments, this is analogous to de Groote’s type

$$\Omega = \text{def } \gamma \rightarrow (\gamma \rightarrow p) \rightarrow p.$$
We now define the type $u$ of updates, also called dynamic propositions, as

$$u = \text{def} \ k \to k = (c \to p) \to c \to p$$

Modulo replacement of $\gamma$ by $c$, and a reversal of the order of arguments, this is analogous to de Groote’s type

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

As with de Groote’s right contexts, intuitively the first (CDP) argument should be thought of as corresponding to the continuation of the discourse.
What is the connection between updates (type $u = \text{def} (c \rightarrow p) \rightarrow (c \rightarrow p)$) and the more usual idea of a dynamic sentence meaning as a context change (type $c \rightarrow c$)?
Updates vs. Context Changes

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

- Every context change $f$ uniquely determines an update by the ‘contraposition’ embedding $\mu : (c \rightarrow c) \rightarrow u$ defined as follows:

$$\mu = \text{def} \ \lambda_{fkc}.(k \ (f \ c))$$
What is the connection between updates (type $u = \text{def} (c \to p) \to (c \to p)$) and the more usual idea of a dynamic sentence meaning as a context change (type $c \to c$)?

Every context change $f$ uniquely determines an update by the ‘contrapositive’ embedding $\mu : (c \to c) \to u$ defined as follows:

$$\mu = \text{def} \lambda f k c. (k (f c))$$

That is, for each context change $f$, the corresponding update $(\mu f)$ maps any CDP $k$ to the CDP $k \circ f = \lambda c. k (f c)$. 
What is the connection between updates (type \( u = \text{def} \ (c \rightarrow p) \rightarrow (c \rightarrow p) \)) and the more usual idea of a dynamic sentence meaning as a context change (type \( c \rightarrow c \))? 

Every context change \( f \) uniquely determines an update by the ‘contrapositive’ embedding \( \mu : (c \rightarrow c) \rightarrow u \) defined as follows:

\[
\mu = \text{def} \lambda f k_c. (k (f c))
\]

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

\[
d_0 = \text{def } u \\
d_{\text{suc } n} = \text{def } \omega \rightarrow d_n
\]
Generalizing Muskens 1996, we treat $n$-ary dynamic properties as functions from $n$ DRs to updates:

$$d_0 = \text{def } u$$

$$d_{(\text{succ } n)} = \text{def } \omega \rightarrow d_n$$

In particular, 0-ary dynamic properties are just updates.
Generalizing Muskens 1996, we treat $n$-ary dynamic properties as functions from $n$ DRs to updates:

$$d_0 = \text{def } u$$

$$d_{(\text{succ}n)} = \text{def } \omega \rightarrow d_n$$

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:

  $$d_0 = \text{def } u$$
  $$d_{\text{suc } n} = \text{def } \omega \rightarrow d_n$$

- 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.
We recursively define the family of *dynamicizer* functions \( \text{dyn}_n : p_n \rightarrow d_n \) as follows:

\[
\text{dyn}_0 = \text{def } \lambda p_k \cdot \lambda c \mid k_{\downarrow c + p} \cdot p \text{ and } (k (c + p)) : p \rightarrow u
\]
We recursively define the family of dynamicizer functions $\text{dyn}_n : p_n \rightarrow d_n$ as follows:

$$\text{dyn}_0 = \text{def } \lambda p_k \cdot \lambda c \mid k \downarrow c + p \cdot p \text{ and } (k (c + p)) : p \rightarrow u$$

$$\text{dyn}_{\text{suc } n} = \text{def } \lambda R_m \cdot (\text{dyn}_n (R [m])) : p_{\text{suc } n} \rightarrow d_{\text{suc } n}$$
Dynamicizing Static Properties

- We recursively define the family of \texttt{dynamicizer} functions \( \texttt{dyn}_n : p_n \rightarrow d_n \) as follows:

  \[
  \texttt{dyn}_0 = \text{def} \lambda_{pk} \cdot \lambda_c \mid k \downarrow_{c+p} \cdot p \text{ and } (k (c + p)) : p \rightarrow u \\
  \texttt{dyn}_{\text{succ} n} = \text{def} \lambda_{Rm} \cdot (\texttt{dyn}_n (R [m])) : p_{\text{succ} n} \rightarrow d_{\text{succ} n}
  \]

- The restriction on the context variable in the definition of \( \texttt{dyn}_0 \) 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.
Examples

\[ \text{RAIN} = \text{def } (\text{dyn}_0 \text{ rain}) = \lambda_{kc}. \text{rain and } (k \ (c + \text{rain})) \]
Examples

\[
\begin{align*}
\text{RAIN} &= \text{def} \ (\text{dyn}_0 \text{ rain}) = \lambda_{kc}. \text{rain} \text{ and } (k \ (c + \text{rain})) \\
\text{DONKEY} &= \text{def} \ (\text{dyn}_1 \text{ donkey}) = \\
&\quad \lambda_{nkc}. (\text{donkey}[n]) \text{ and } (k \ (c + (\text{donkey}[n])))
\end{align*}
\]
Some Dynamic Properties

- Examples

RAIN = def (dyn_0 rain) = \lambda_{kc}.\text{rain and } (k \(c + \text{rain}))

DONKEY = def (dyn_1 donkey) = \lambda_{nkc}.(\text{donkey}[n]) \text{ and } (k \(c + (\text{donkey}[n])))

OWN = def (dyn_2 own) = \lambda_{mnkc}.(\text{own}[m][n]) \text{ and } (k \(c + (\text{own}[m][n])))
Some Dynamic Properties

- **Examples**

  \[
  \text{RAIN} = \text{def} \ (\text{dyn}_0 \ \text{rain}) = \lambda_{kc}.\text{rain} \ \text{and} \ (k \ (c + \text{rain}))
  \]

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

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

- **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, \textit{dynamic conjunction} is just composition of updates:

\[
\text{AND} \ = \ \text{def} \ \lambda_{u v k}.u(vk)
\]

\textbf{Example:}

It rains. It snows. \(\Rightarrow\) rain and snow \(=\) \(\lambda_{k}.(\text{dyn}0\text{rain})(\text{dyn}0\text{snow})k\) \(=\) \(\lambda_{k}.(\lambda_{c}k.\text{rain and } (k(c + \text{rain}))))(\lambda_{c}k.\text{snow and } (k(c + \text{snow}))))\)

Note that the CG of the context \((c + \text{rain})\) passed to the second conjunct \((\text{snow})\) contains the static propositional content \((\text{rain})\) 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.
As usual in this line of work, **dynamic conjunction** is just composition of updates:

\[
\text{AND} = \text{def } \lambda_{u,v,k}.u\,(v\,k)
\]

Example: *It rains. It snows.* $\rightsquigarrow$

RAIN AND SNOW $= \lambda_k.(\text{dyn}_0\text{ rain})(\text{dyn}_0\text{ snow})\,k$

$= \lambda_k.(\lambda_{k_c}.\text{rain and } (k\,(c + \text{ rain}))) (\lambda_c.\text{snow and } (k\,(c + \text{ snow})))$

$= \lambda_{k_c}.\text{rain and snow and } (k\,(c + \text{ rain + snow})))$
As usual in this line of work, **dynamic conjunction** is just composition of updates:

\[
\text{AND} \triangleq \lambda_{uvk}.u(vk)
\]

**Example:** \textit{It rains. It snows.} \implies
\[
\text{RAIN AND SNOW} = \lambda_k.((\text{dyn}_0 \text{ rain})(\text{dyn}_0 \text{ snow}) k)
\]
\[
= \lambda_k.(\lambda_{kc}.\text{rain and } (k(c + \text{rain}))) (\lambda_c.\text{snow and } (k(c + \text{snow})))
\]
\[
= \lambda_{kc}.\text{rain and snow and } (k(c + \text{rain} + \text{snow}))
\]

Note that the CG of the context \((c + \text{rain})\) passed to the second conjunct \((\text{SNOW})\) contains the static propositional content \((\text{rain})\) of the first conjunct.
Dynamic Conjunction

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

  \[
  \text{AND} = \text{def} \lambda_{uvk}.u(vk)
  \]

- Example: *It rains. It snows.* \(\rightarrow\)

  \[
  \text{RAIN AND SNOW} = \lambda_k.(\text{dyn}_0 \text{rain})((\text{dyn}_0 \text{snow}) k)
  \]

  \[
  = \lambda_k.((\lambda_{kc}.\text{rain and } (k(c + \text{rain}))))(\lambda_c.\text{snow and } (k(c + \text{snow}))))
  \]

  \[
  = \lambda_{kc}.\text{rain and snow and } (k(c + \text{rain + snow}))
  \]

- Note that the CG of the context \((c + \text{rain})\) passed to the second conjunct \((\text{SNOW})\) contains the static propositional content \((\text{rain})\) 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.*
The dynamic existential quantifier is defined as follows:

\[
\text{EXISTS} = \text{def} \ \lambda_{Dkc}.\exists x. D (\text{next } c) k (c :: x) : d \to u
\]
The **dynamic existential quantifier** is defined as follows:

\[
\text{EXISTS} = \text{def } \lambda_{Dkc} \cdot \exists \lambda_x. D(\text{next } c) k(c :: x) : d \to u
\]

Note that what is quantified over is possible anchors for the newly introduced DR.
The **dynamic existential quantifier** is defined as follows:

\[
\text{EXISTS} = \text{def } \lambda_{D,k,c} \cdot \exists x. D (\text{next } c) k (c :: x) : d \rightarrow u
\]

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:

\[
A = \text{def } \lambda_{D,E} \cdot \text{EXISTS } \lambda_n. (D \ n) \text{ AND } (E \ n) : d \rightarrow d \rightarrow u
\]

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

\[ a \text{ donkey enters } \sim A \text{ DONKEY ENTER} \]
\[ = \exists \lambda_n.(\text{DONKEY } n) \text{ AND } (\text{ENTER } n) \]
\[ = \lambda_{kc} \exists \lambda_x.((\text{DONKEY } (c)) \text{ AND } (\text{ENTER } (c))) k (c :: x) \]
\[ = \lambda_{kc} \exists \lambda_x.(\text{donkey } x) \text{ and } (\text{enter } x) \]
\[ \text{and } (k ((c :: x) + \text{donkey } x + \text{enter } x)) \]
Example with an Indefinite

\[ a \text{ donkey enters } \leadsto \text{A DONKEY ENTER} \]

\[ = \text{exists } \lambda_n. (\text{DONKEY } n) \text{ AND (ENTER } n) \]

\[ = \lambda_{kc}. \text{exists } \lambda_x. ((\text{DONKEY } (c)) \text{ AND (ENTER } (c))) k (c :: x) \]

\[ = \lambda_{kc}. \text{exists } \lambda_x. (\text{donkey } x) \text{ and (enter } x) \]

\[ \text{and (} k ((c :: x) + \text{donkey } x + \text{enter } x))) \]

For each possible choice of anchor \( c \) for the new DR \((\text{next } c)\), the propositions \((\text{donkey } x)\) and \((\text{enter } x)\) are included in the context that gets passed to the rest of the discourse.
Example with an Indefinite

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

- For each possible choice of anchor \( c \) for the new DR \( \text{next } c \), the propositions \( \text{donkey } x \) and \( \text{enter } x \) 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 \textit{the donkey that entered}. 
Functions for Handling Presupposition

- the definedness check ↓
- the staticizer function stat
- the definiteness function def
The Definedness Check

- The **definedness** function

  \[ \downarrow = \text{def } \lambda_{kc}.(\text{dom } k \ c) : k \rightarrow c \rightarrow t \]

  (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
  \[
  \downarrow \overset{\text{def}}{=} \lambda_{kc}.(\text{dom } k \ c) : k \rightarrow c \rightarrow t
  \]
  (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

\[ \top = \text{def } \lambda c. \text{true} : k \]

corresponds intuitively to the end of the discourse.
The Staticizer Function

- The trivial CDP

\[ \top = \text{def } \lambda_c.\text{true} : k \]

corresponds intuitively to the end of the discourse.

- The staticizer function

\[ \text{stat} = \text{def } \lambda_{cu}.(u \top c) : c \rightarrow u \rightarrow p \]

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

\[ \top = \text{def } \lambda c. \text{true} : k \]

corresponds intuitively to the end of the discourse.

- The staticizer function

\[ \text{stat} = \text{def } \lambda c u. (u \top c) : c \to u \rightarrow p \]

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):

\[ \vdash \forall c \forall p. (\text{stat } c (\text{dyn}_0 p)) \equiv p \]

where \( \equiv \) is truth-conditional equivalence (mutual entailment).
The Definiteness Function

For each \( n \), the **definiteness** function

\[
def_n = \lambda_c D \cdot \bigcup_{r \in c} \lambda_i : \omega_n \cdot (p c) \text{ entails } (\text{stat } c (D i)) : c_n \to d \to \omega_n
\]

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

$$\text{def}_n = \lambda_{cD} \cdot \bigcup_{(r_c)} \lambda_i: \omega_n \cdot (p \ c) \text{ entails } (\text{stat } c \ (D \ i)) : c_n \rightarrow d \rightarrow \omega_n$$

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:

$$\text{IT} = \lambda_{Dkc} \cdot D \ (\text{def } c \ \text{NONHUMAN}) \ k \ c : d \rightarrow u$$

where NONHUMAN $= \lambda_{c} \cdot (\text{dyn}_1 \ \text{nonhuman})$
The Definiteness Function

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

$$\text{def}_n = \lambda_{cD} \cdot \bigcup_{(r_c)} \lambda_{i: \omega_n} \cdot (p_c) \text{ entails } (\text{stat } c (D \ i)) : c_n \rightarrow d \rightarrow \omega_n$$

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:

$$\text{IT} = \lambda_{Dkc} \cdot D (\text{def } c \ \text{NONHUMAN}) \ k \ c : d \rightarrow u$$

where NONHUMAN = $\lambda_{\text{nonhuman}}$

- *It brays.* $\Rightarrow$ (IT BRAY) =

$$\lambda_{kc} \cdot \text{bray}[\text{def } c \ \text{NONHUMAN}]$$

and $(k (c + \text{bray}[\text{def } c \ \text{NONHUMAN}]))$
A Definite Anaphora Example

- A donkey enters. It brays. \( \rightsquigarrow \)
A donkey enters. It brays. \[ A \text{ donkey enters. It brays. } \rightarrow \]

\[
\begin{align*}
(A \text{ DONKEY ENTER}) \text{ AND } (IT \text{ BRAY}) &= \\
\lambda_k c. \exists \lambda x. (\text{donkey } x) \text{ and } (\text{enter } x) \\
\text{and } \text{bray}[\text{def } c'[c, x] \text{ NONUMAN}] \\
\text{and } (k (c'[c, x] + \text{bray}[\text{def } c'[c, x] \text{ NONHUMAN}]))
\end{align*}
\]

where \( c'[c, x] \) is \( (c :: x) + \text{donkey } x + \text{enter } x \).
A Definite Anaphora Example

- *A donkey enters. It brays.*

\[
\begin{align*}
(A \text{ DONKEY ENTER}) \text{ AND } (IT \text{ Bray}) = \\
\lambda_{kc}.\exists x.(\text{donkey } x) \text{ and (enter } x) \\
\text{ and bray[def } c'[c, x] \text{ NONUMAN]} \\
\text{ and } (k (c'[c, x] + \text{bray[def } c'[c, x]\text{ NONHUMAN})))
\end{align*}
\]

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

- As long as (1) it is in the CG that donkeys are nonhuman, and (2) no inferrably nonhuman DR more salient than (\text{next } c) is present, then \([\text{def } c'[c, x] \text{ NONHUMAN}]) = x\).

- 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.
  6. 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:

\[
\text{NOT} = \text{def} \; \lambda_{ukc}.\text{dyn}_0( \text{not} (\text{stat } c \; u)) \quad k \; c : u \rightarrow u
\]
Dynamic Negation

- Analogizing directly from de Groote would give the following dynamic meaning:
  \[ \text{NOT} = \text{def} \quad \lambda_{ukc}.\text{dyn}_0(\text{not (stat c u)}) \quad k \colon c : u \rightarrow u \]

- We amend this to:
  \[ \lambda_{uk}.\lambda_c | (u k)_{c} \cdot \text{dyn}_0(\text{not (stat c u)}) \quad k \colon c \]

- The restriction on the context variable requires that the denial be defined in the same contexts that the update being denied is defined.
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.
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.
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.
A naive dynamic meaning for *suck* would be:

\[
\text{SUCK} = \text{def } \lambda_{u k c}. \text{dyn}_0(\text{suck}(\text{stat} c u)) k c : u \to u
\]
A naive dynamic meaning for *suck* would be:

\[
\text{SUCK} \equiv \lambda_{ukc}. \text{dyn}_0(\text{suck}(\text{stat } c \ u)) \ k \ c : u \to u
\]

We amend this to:

\[
\lambda_{uk} \cdot \lambda_c | (p \ c) \text{ entails } (\text{stat } c \ u). \text{dyn}_0(\text{suck}(\text{stat } c \ u)) \ k \ c
\]
A naive dynamic meaning for *suck* would be:

\[ \text{SUCK} = \text{def} \lambda_{ukc} . \text{dyn}_0 (\text{suck} (\text{stat} \ c \ u)) \ k \ c : u \to u \]

We amend this to:

\[ \lambda_{uk} \cdot \lambda_{c} \mid (p \ c) \text{ entails } (\text{stat} \ c \ u) . \text{dyn}_0 (\text{suck} (\text{stat} \ c \ u)) \ k \ c \]

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 nor 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.
We start with de Groote’s dynamic conditional semantics:

$$\text{IF} = \text{def} \lambda_{uv}.\text{NOT} \ (u \ \text{AND} \ (\text{NOT} \ v)) : u \rightarrow u \rightarrow u$$
We start with de Groote’s dynamic conditional semantics:

\[ IF = \text{def } \lambda_{uv}. \neg (u \land \neg v) : u \rightarrow u \rightarrow u \]

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:

\[ \vdash IF \; u \; v = \lambda_{kc}. \neg (p \land \neg q) \equiv \lambda_{kc}. p \text{ implies } q \]
We start with de Groote’s dynamic conditional semantics:

\[ \text{IF} = \text{def} \lambda_{uv}.\text{NOT} (u \text{ AND } (\text{NOT } v)) : u \rightarrow u \rightarrow u \]

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:

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

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.
We start with de Groote’s dynamic conditional semantics:

\[
\text{IF} = \text{def} \lambda_{uv}.\text{NOT} (u \text{ AND } (\text{NOT} v)) : u \rightarrow u \rightarrow u
\]

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:

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

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

\[ \lambda_{uvk} \cdot \lambda_c \mid (\text{stat } c \ u) \text{ indep } (p \ c) \cdot \neg (u \text{ AND } (\neg v)) \ \forall c \]

where indep : p \rightarrow p \rightarrow t is defined as:

\[ \lambda_{pq}.\neg((p \text{ entails } q) \lor (p \text{ entails } (\neg q))) \]

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