# A logic of meaning and synonymy

Fritz Hamm and Yiannis Moschovakis
Universität Tübingen, UCLA and University of Athens

2010 ESSLLI, Copenhagen

# The development of Mathematical Logic

...(comic book version)

(1) Language (Frege 1879, Hilbert school).
First Order Language FOL is chosen as the most suitable formal language: sufficiently rich so that mathematical theories can be expressed in it and sufficiently simple to be profitably studied with mathematical methods.

(2) Interpretations (Tarski).
A precise (set theoretic) interpretation of FOL is given in first-order structures — Tarski's definition of satisfaction and truth.

(3) Proof theory (Hilbert school).
Precise (set theoretic) specification of proof systems for FOL.

(4) The Completeness Theorem (Gödel 1928).
Identification of the provable FOL sentences with those which are valid (true in all first-order structures).

- Answers the question of what follows from what by logic alone

# Frege's sense and denotation

- $1+1=2$ vs. *there are infinitely many prime numbers*

  Same truth value but different thoughts are expressed

- $A \longrightarrow \ \mathit{sense}(A) \longrightarrow \ \mathit{den}(A)$

- Terms denote objects and include sentences, which denote either 1 (truth) or 0 (falsity).

- The sense (meaning) of a term "contains the mode of presentation of the denotation".

- The function $A \longrightarrow \ \mathit{sense}(A)$ is compositional.

# Frege on sense (which he did not define)

"*[the sense of a sign] may be the common property of many people*" Meanings are public (abstract?) objects

"*The sense of a proper name is grasped by everyone who is sufficiently familiar with the language . . . Comprehensive knowledge of the thing denoted . . . we never attain*"

Speakers of the language know the meanings of terms

"*The same sense has different expressions in different languages or even in the same language*"

"*The difference between a translation and the original text should properly not overstep the [level of the idea]*"

Faithful translation should preserve meaning

sense($A$) $\sim$ the part of the semantic value of $A$ which is preserved under faithful translation (the elephant in the room)

# A logic of meaning and synonymy (simplified, all lies are white)

(1) Language. The *typed $\lambda$-calculus with acyclic recursion* $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, an extension of Richard Montague's *language of intensional logic*.

(2) Interpretation. In every suitable *higher type structure* $\mathfrak{M}$, each *closed term* $A$ of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ is assigned:

$$\boxed{\text{a value } \mathrm{den}^{\mathfrak{M}}(A) \quad \textbf{and} \quad \text{a referential intension } \mathrm{int}^{\mathfrak{M}}(A)}$$

$\mathrm{int}^{\mathfrak{M}}(A)$ models the meaning of $A$ and determines $\mathrm{den}^{\mathfrak{M}}(A)$

$$\boxed{\begin{array}{l} A \approx^{\mathfrak{M}} B \quad (A \text{ is synonymous with } B \text{ in } \mathfrak{M}) \\ \qquad\qquad \iff \mathrm{int}^{\mathfrak{M}}(A) \cong \mathrm{int}^{\mathfrak{M}}(B) \quad (\text{naturally isomorphic, } =) \end{array}}$$

(3) The Reduction Calculus of meaning and synonymy:

$$\boxed{\begin{array}{l} A \Rightarrow B \iff A \approx_{\ell} B \quad (\text{synonymous in all structures}) \\ \text{and } B \text{ expresses } \mathrm{int}(A) = \mathrm{int}(B) \text{ more (no less) directly than } A. \end{array}}$$

(4) Completeness. There are decidable and complete axiomatizations of (global) denotational identity and synonymy

# What is the referential intension of a term $A$?

- As a slogan: $\boxed{\mathrm{int}(A) \textit{ is the algorithm which computes } \mathrm{den}(A)}$
- *The meaning of a term A is faithfully represented by an (abstract) procedure which computes its denotation* $\mathrm{den}(A)$.
- (1) If you know the meaning of $A$, then you have (in principle) a method for determining its denotation.

  (2) If you have a method for determining the denotation of $A$, then you know the meaning of $A$.
- It has been argued that neither of these principles can be found in Frege; and it has also been argued that this is exactly what Frege means when he says

"*The sense contains the mode of presentation of the denotation*"

- The theory of referential intensions imports ideas from the theory of programming languages that go beyond modelling meanings by algorithms

# Outline

Introduction (already done)

1. Some remarks on the methodology we will follow
2. Syntax and denotational semantics of $L^{\lambda}_{ar}$
3. Examples from natural language (also throughout)
4. Overview of referential intension theory
5. The reduction calculus
6. Referential intensions; referential and logical synonymy
7. Propositional attitudes

Afterword

*Sense and denotation as algorithm and value* (1994)
*A logical calculus of meaning and synonymy* (2006)
*Two aspects of situated meaning* (with E. Kalyvianaki (2008))
Posted in www.math.ucla.edu/$\sim$ynm

# Rendering (of natural language into $L^{\lambda}_{ar}$)

The rigorous logical analysis of a phrase from natural language will start by rendering (translating) it into the formal language $L^{\lambda}_{ar}$.

every man loves some woman

$$\xrightarrow{\text{render}} \text{every}(\text{man})\Big[\lambda(u)\Big(\text{some}(\text{woman})(\lambda(v)\text{loves}(u,v))\Big)\Big]$$

coordination:

Abelard loved and honored Eloise

$$\xrightarrow{\text{render}} \lambda(u,v)\Big(\text{loved}(u,v) \text{ and } \text{honored}(u,v)\Big)(\text{Abelard}, \text{Eloise})$$

coindexing:

Abelard loved Eloise and (he) honored her

$$\xrightarrow{\text{render}} \text{loved}(\dot{a},\dot{e}) \text{ and } \text{honored}(\dot{a},\dot{e}) \text{ where } \{\dot{a} := \text{Abelard}, \dot{e} := \text{Eloise}\}$$

$\boxed{\text{natural language expression}}$ + informal context

$$\xrightarrow{\text{render}} \boxed{\text{formal } L^{\lambda}_{ar} \text{ term}} + \text{state}$$

# Is all language situated?

$$\text{He loves her} \xrightarrow{\text{render}} A \equiv \text{loves}(\text{he}, \text{her})$$

- The truth and meaning of $A$ depend on who "he" and "her" are, when the utterance was made, etc.
  These are all determined by the informal context and coded in the state
- In Montague's LIL, every term $A$ which expresses a sentence of natural language is interpreted by its Carnap intension

$$\text{CI}(A) : \text{States} \to \text{Truth values}$$

  a function which assigns a truth value to every state.
- Every term of LIL is interpreted by a function on the set of states.

Slogan: $\boxed{\textit{All language is situated}}$

- In LIL $\text{den}(3 + 2 = 5)$ is the constant function $(a \mapsto \text{true})$
  ...which has a different *logical status* (type) from the object "true"
- We will not adopt the slogan (will allow variables over states, etc.)

# Propositional attitudes

Peter declares that he loves John's sister

$$\xrightarrow{\text{formalize}} \mathrm{Decl.}(\mathrm{Peter}, \mathrm{loves}(\mathrm{he}, \mathrm{sister}(\mathrm{John})))$$

$$\xrightarrow{\text{coindex}} A \equiv \mathrm{Decl.}(\dot{p}, \mathrm{loves}(\dot{p}, \mathrm{sister}(\mathrm{John}))) \text{ where } \{\dot{p} := \mathrm{Peter}\}$$

$$\equiv \mathrm{Decl.}(\dot{p}, L) \text{ where } \{\dot{p} := \mathrm{Peter}\}$$

where

$$L \equiv \mathrm{loves}(\dot{p}, \mathrm{sister}(\mathrm{John}))$$

- *The truth and meaning of A depend on the meaning of L (for a fixed value of $\dot{p}$), not just its truth value*
- Other attitudinal constants like Decl. include

$$\text{Claims that} \ldots, \quad \text{Says that} \ldots, \quad \text{Believes that} \ldots$$

- We will first develop the theory of referential intensions for the denotational part of the language
and then interpret the full language into its denotational part

## The $\lambda$-calculus with acyclic recursion $\mathsf{L}^{\lambda}_{\mathrm{ar}}$: types

**Basic types** Entities : e Truth values : t States : s

$$\boxed{\sigma :\equiv \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid (\sigma_1 \to \sigma_2)}$$

**Interpretations** (standard)

$$\begin{aligned} \mathbb{T}_{\mathsf{e}} &= \text{a given set (or class) of people, objects, etc.} \\ \mathbb{T}_{\mathsf{s}} &= \text{a given set of states} \\ \{0, 1, er\} \subseteq \mathbb{T}_{\mathsf{t}} &= \text{a given set of truth values } \subseteq \mathbb{T}_{\mathsf{e}} \end{aligned}$$

$$\boxed{\mathbb{T}_{(\sigma\to\tau)} = (\mathbb{T}_\sigma \to \mathbb{T}_\tau) = \text{the set of all functions } p : \mathbb{T}_\sigma \to \mathbb{T}_\tau}$$

**State** $a = (\mathrm{world}(a), \mathrm{time}(a), \mathrm{location}(a), \mathrm{agent\ (speaker)}(a), \delta)$

$$\delta(\mathrm{He}_1) = \ldots, \quad \delta(\mathrm{this}) = \ldots, \text{ etc.}$$

$$er = \text{ error } \quad \Big(\mathrm{den}(\text{the King of France is bald}(a) = er\Big)$$

$$\boxed{x : \sigma \iff x \in \mathbb{T}_\sigma \quad (x \text{ is an object of type } \sigma)}$$

# Special kinds of types and objects

**Pure types** $\boxed{\sigma :\equiv \mathsf{e} \mid \mathsf{t} \mid (\sigma_1 \to \sigma_2)}$

$\tilde{\mathsf{t}} :\equiv (\mathsf{s} \to \mathsf{t})$ (Carnap intensions)

$\tilde{\mathsf{e}} :\equiv (\mathsf{s} \to \mathsf{e})$ (state-dependent entities (individual concepts))

**Natural language types** $\boxed{\sigma :\equiv \tilde{\mathsf{e}} \mid \tilde{\mathsf{t}} \mid (\sigma_1 \to \sigma_2)}$

- *The terms which are rendered by natural language phrases are of natural language type* (True?)

**State-dependent unary quantifier types** (every(boy)):

$$\tilde{\mathsf{q}} :\equiv ((\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{t}})$$

**Abbreviations** $\sigma_1 \times \sigma_2 \to \tau :\equiv (\sigma_1 \to (\sigma_2 \to \tau))$

# Constants; the lexicon

**Empirical (denotational) constants:**

| | | |
|---|---|---|
| Entities | 0, 1, 2,. . . , *er*: | e |
| Names, demonstratives | John, I, he, him, today: | $\tilde{e}$ |
| Common nouns | man, unicorn, temperature: | $\tilde{e} \to \tilde{t}$ |
| Adjectives | tall, young: | $(\tilde{e} \to \tilde{t}) \to (\tilde{e} \to \tilde{t})$ |
| Propositions | it rains: | $\tilde{t}$ |
| Intransitive verbs | stand, run, rise: | $\tilde{e} \to \tilde{t}$ |
| Transitive verbs | find, love, be, seek: | $(\tilde{e} \times \tilde{e}) \to \tilde{t}$ |
| Adverbs | rapidly, allegedly: | $(\tilde{e} \to \tilde{t}) \to (\tilde{e} \to \tilde{t})$ |

**Logical constants:**

$$\begin{array}{rcl} =_\sigma & : & \sigma \times \sigma \to \mathrm{t} \\ \neg & : & \mathrm{t} \to \mathrm{t} \\ \&, \vee, \Rightarrow & : & \mathrm{t} \times \mathrm{t} \to \mathrm{t} \\ \forall_\sigma, \exists_\sigma & : & (\sigma \to \mathrm{t}) \to \mathrm{t} \end{array} \qquad \begin{array}{rcl} \text{not}, \Box, \text{in the future} & : & \tilde{\mathrm{t}} \to \tilde{\mathrm{t}} \\ \text{and}, \text{or}, \text{if .. then ..} & : & \tilde{\mathrm{t}} \times \tilde{\mathrm{t}} \to \tilde{\mathrm{t}} \\ \text{every}, \text{some} & : & (\tilde{\mathrm{e}} \to \tilde{\mathrm{t}}) \to \tilde{\mathrm{q}} \\ \text{the} & : & (\tilde{\mathrm{e}} \to \tilde{\mathrm{t}}) \to \tilde{\mathrm{e}} \end{array}$$

- $\boxed{\textit{A set } K \textit{ of typed constants determines the language } \mathrm{L}^{\lambda}_{\mathrm{ar}}(K)}$

# Typed variables

**Two kinds of (typed) variables**

- Pure variables of type $\sigma$: $v_0^\sigma, v_1^\sigma \dots$
- Recursion variables or locations of type $\sigma$: $\dot{v}_0^\sigma, \dot{v}_1^\sigma \dots$

- Both $v_i^\sigma$ and $\dot{v}_i^\sigma$ will be interpreted by arbitrary objects $x : \sigma$
  ... but they will be treated differently in the syntax
- Pure variables will be bound by the $\lambda$-operator (as in the typed $\lambda$-calculus)
- Locations will be used to make (formal) assignments

$$\dot{p} := A$$

and will be bound by the recursion construct $\boxed{\text{where}}$

# Interpretations of $\mathsf{L}^{\lambda}_{\mathsf{ar}}(K)$

- A (standard) structure

$$\mathfrak{M} = (\mathbb{T}_{\mathsf{e}}, \mathbb{T}_{\mathsf{s}}, \mathbb{T}_{\mathsf{t}}, \{\overline{c}\}_{c \in K})$$

  for $\mathsf{L}^{\lambda}_{\mathsf{ar}}(K)$ is specified by given sets of basic types and a given denotation (value) $\overline{c}$ for each constant $c \in K$.

- There is a fixed structure

$$\mathfrak{M}_0 = \text{our universe}$$

- A valuation (assignment) in $\mathfrak{M}$ is any function g which assigns to each variable $x$ of type $\sigma$ (of either kind) an object $\mathrm{g}(x) : \sigma$.

# Terms

$$\boxed{A :\equiv x \mid \mathsf{c} \mid A(B) \mid \lambda(u)(A) \mid A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}}$$

- Recursive definition, starting with the variables and the constants
- Three formations rules:
  application, $\lambda$-abstraction and acyclic recursion
- Conditions for each of the three formation rules to apply
- Each term is assigned a type $\boxed{A : \sigma \iff A \text{ is a term of type } \sigma}$
- $\mathrm{FV}(A) =$ the set of free occurrences of variables in $A$
- $\mathrm{den}(A)^{\mathfrak{M}}(\mathrm{g}) =$ the denotation of $A$ for the valuation g in $\mathfrak{M}$
  (We will skip the superscript $\mathfrak{M}$ in the definitions below)

# Abbreviations, congruence, formal replacement

**Abbreviations and misspellings**:

$$A(B)(C) \equiv A(B, C),$$
$$A[B(C, D)] \equiv A(B(C)(D)),$$
$$A \text{ where } \{\ \} \equiv A, etc.$$

**Term Congruence**: $A \equiv_c B$ is an equivalence relation on terms such that

- $A \equiv_c B$ if $B$ is constructed from $A$ by alphabetic changes of bound variables and
- $A \text{ where } \{\dot{p} := B, \dot{q} := C\} \equiv_c A \text{ where } \{\dot{q} := C, \dot{p} := B\}$

**Term replacement**:

$A\{x :\equiv B\}$ = the result of replacing every free occurrence of the variable $x$ in $A$ by the term $B$

- Free if no free variable of $B$ is bound in $A\{x :\equiv B\}$

# Terms: constants and variables

$$\boxed{A :\equiv \underbrace{x \mid \mathsf{c}} \mid A(B) \mid \lambda(u)(A) \mid A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}}$$

(T0) *If $x$ is a variable of type $\sigma$ of either kind, then*

$$x : \sigma, \quad \mathrm{FV}(x) = \{x\}, \quad \mathrm{den}(x)(\mathrm{g}) = \mathrm{g}(x)$$

*If c is a constant of type $\sigma$, then*

$$\mathsf{c} : \sigma, \quad \mathrm{FV}(\mathsf{c}) = \emptyset, \quad \mathrm{den}(\mathsf{c})(\mathrm{g}) = \overline{c}$$

Examples:

$$\text{George, He} : \tilde{\mathrm{e}}, \quad \text{runs, man} : \tilde{\mathrm{e}} \to \tilde{\mathrm{t}}, \quad \text{loves} : \tilde{\mathrm{e}} \times \tilde{\mathrm{e}} \to \tilde{\mathrm{t}}$$

# Terms: application

$$\boxed{A :\equiv x \mid \mathsf{c} \mid \underbrace{A(B)} \mid \lambda(u)(A) \mid A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}}$$

(T1) *If $A : (\sigma \to \tau)$ and $B : \sigma$, then*

$$A(B) : \tau, \qquad \mathrm{FV}(A(B)) = \mathrm{FV}(A) \cup \mathrm{FV}(B),$$
$$\mathrm{den}(A(B))(\mathrm{g}) = \mathrm{den}(A)(\mathrm{g})(\mathrm{den}(B)(\mathrm{g}))$$

Examples (predication, quantification, etc.)

$$\text{George runs} \xrightarrow{\text{render}} \mathsf{runs}(\mathsf{George}) : \tilde{\mathrm{t}}$$

$$\begin{aligned} \text{Abelard loved Eloise} &\xrightarrow{\text{render}} \mathsf{loved}(\mathsf{Abelard})(\mathsf{Eloise}) \\ &\equiv \mathsf{loved}(\mathsf{Abelard}, \mathsf{Eloise}) : \tilde{\mathrm{t}} \end{aligned}$$

$$\text{Every man} \xrightarrow{\text{render}} \mathsf{every}(\mathsf{man}) : (\tilde{\mathrm{e}} \to \tilde{\mathrm{t}}) \to \tilde{\mathrm{t}}$$

$$\begin{aligned} \text{Every man dies} &\xrightarrow{\text{render}} \mathsf{every}(\mathsf{man})(\mathsf{dies}) \\ &\equiv \mathsf{every}(\mathsf{man}, \mathsf{dies}) : \tilde{\mathrm{t}} \end{aligned}$$

# Terms: $\lambda$-abstraction

$$\boxed{A :\equiv x \mid \mathsf{c} \mid A(B) \mid \underbrace{\lambda(u)(A)} \mid A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}}$$

(T2) *If $A : \tau$ and $u$ is a pure variable of type $\sigma$, then*

$$\lambda(u)(A) : (\sigma \to \tau), \quad \mathrm{FV}(\lambda(u)(A)) = \mathrm{FV}(A) \setminus \{u\}$$
$$\mathrm{den}(\lambda(u)(A))(\mathrm{g}) = h : \mathbb{T}_\sigma \to \mathbb{T}_\tau$$
$$\text{such that } h(x) = \mathrm{den}(A)(\mathrm{g}\{u := x\})$$

($\mathrm{g}\{u := x\}$ is the update of g by the assignment $u := x$)

| |
|---|
| Example (coordination):<br><br>Abelard loved and honored Eloise<br><br>$\xrightarrow{\text{render}} \lambda(u,v)\Big(\text{loved}(u,v) \text{ and honored}(u,v)\Big)(\text{Abelard}, \text{Eloise})$ |

# Terms: acyclic recursion

$$\boxed{A :\equiv x \mid \mathsf{c} \mid A(B) \mid \lambda(u)(A) \mid \underbrace{A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}}}$$

An acyclic system is a sequence of term assignments

$$\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\} \quad (\mathrm{type}(\dot{p}_i) = \mathrm{type}(A_i))$$

to the distinct locations $\dot{p}_1, \dots, \dot{p}_n$, such that for suitable numbers $\mathrm{rank}(\dot{p}_1), \dots, \mathrm{rank}(\dot{p}_n)$,

$$\text{if } \dot{p}_j \text{ occurs free in } A_i \text{, then } \mathrm{rank}(\dot{p}_j) < \mathrm{rank}(\dot{p}_i)$$

(T3) *If* $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$ *is an acyclic system and* $A_0 : \sigma$, *then*

$$A \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\} : \sigma,$$

$$\mathrm{FV}(A) = \mathrm{FV}(A_0) \cup \mathrm{FV}(A_1) \cup \cdots \cup \mathrm{FV}(A_n) \setminus \{\dot{p}_1, \dots, \dot{p}_n\},$$

$$\mathrm{den}(A)(\mathrm{g}) = \mathrm{den}(A_0)(\mathrm{g}\{\dot{p}_1 := \overline{p}_1, \dots, \dot{p}_n := \overline{p}_n\})$$

where $\overline{p}_1, \dots, \overline{p}_n$ are the unique solutions of the system

$$\overline{p}_i = \mathrm{den}(A_i)(\mathrm{g}\{\dot{p}_1 := \overline{p}_1, \dots, \dot{p}_n := \overline{p}_n\}) \quad (i = 1, \dots, n)$$

# John loves Mary and dislikes her husband

$$A \equiv \dot{p} \text{ and } \dot{q} \text{ where } \{\dot{p} := \text{loves}(j, \dot{m}), \dot{q} := \text{dislikes}(j, \dot{h}),$$
$$\dot{h} := \text{husband}(\dot{m}), j := \text{John}, \dot{m} := \text{Mary}\} : \tilde{\mathrm{t}}$$

Stage 1: $\overline{\jmath} := \text{John} : \tilde{\mathrm{e}}$, $\overline{m} := \text{Mary} : \tilde{\mathrm{e}}$
Stage 2: $\overline{h} := \text{husband}(\overline{m}) = \text{Mary's husband} : \tilde{\mathrm{e}}$
$\overline{p} := \text{loves}(\overline{\jmath}, \overline{m}) : \tilde{\mathrm{t}}$
Stage 3: $\overline{q} := \text{dislikes}(\overline{\jmath}, \overline{h}) : \tilde{\mathrm{t}}$
Stage 4: $\text{den}(A) = \overline{p} \text{ and } \overline{q} : \tilde{\mathrm{t}}$

For every state $a$,

$$\text{den}(A)(a) = (\overline{p} \text{ and } \overline{q})(a) = \overline{p}(a) \text{ and } \overline{q}(a)$$

= the truth value of "John loves Mary and dislikes her husband" in state $a$

(= *er* if Mary does not have exactly one husband in state $a$)

# The logic of denotations for $\mathsf{L}^{\lambda}_{\mathrm{ar}}$

*There are many interpretations for* $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$*, some non-standard*

$$\boxed{\mathfrak{M}_0 = \text{ a specific standard interpretation (our universe)}}$$

$$\begin{aligned}
\mathfrak{M}, \mathrm{g} \models A = B &\iff \mathrm{den}(A)(\mathrm{g}) = \mathrm{den}(B)(\mathrm{g}) \text{ in } \mathfrak{M} \\
\mathfrak{M} \models A = B &\iff \text{for all g, } \mathfrak{M}, \mathrm{g} \models A = B \\
\models A = B &\iff \text{ for every } \mathfrak{M},\ \mathfrak{M} \models A = B
\end{aligned}$$

The key tool for establishing denotational identities is

$$\models \Big(\lambda(u)A\Big)(B) = A\{u :\equiv B\} \qquad\qquad (\beta\text{-conversion})$$

**Theorem** (from classical results about the typed $\lambda$-calculus)

*There is a complete and decidable axiomatization of* $\models A = B$

we only use this via: $\boxed{\text{if } \models A = B, \text{ then } \mathfrak{M}_0 \models A = B}$

Caution! $\beta$-**conversion does not preserve meaning**

# Descriptions

$$\mathrm{the}(p)(a) = \begin{cases} \text{the unique } y \in \mathbb{T}_\mathsf{e} \text{ such that } p(b \mapsto y, a), & \text{if it exists,} \\ er, & \text{otherwise,} \end{cases}$$

where $b \mapsto y$ is the constant function on the states with value $y$.

$$\begin{aligned} \text{Mary's husband} &\xrightarrow{\text{render}} \mathrm{the}(\lambda(x)\mathrm{married}(x, \mathrm{Mary})) \\ &\qquad\qquad \equiv \mathrm{husband}(\mathrm{Mary}) : \tilde{\mathrm{e}} \\ \text{Mary's husband is tall} &\xrightarrow{\text{render}} \mathrm{tall}(\mathrm{man})(\mathrm{husband}(\mathrm{Mary})) : \tilde{\mathrm{t}} \end{aligned}$$

$$\begin{aligned} &\mathrm{den}(\mathrm{tall}(\mathrm{man})(\mathrm{husband}(\mathrm{Mary})))(a) \\ &= \begin{cases} 1, & \text{if Mary's husband in state } a \text{ is tall among men in state } a \\ 0, & \text{if Mary's husband in state } a \text{ is not tall among men in state } a \\ er, & \text{if Mary does not have a unique husband in state } a \end{cases} \end{aligned}$$

# Errors and presuppositions

- We use *er* to model simple, logical presuppositions
- A proposition $P$ is a logical presupposition of a term $A$ if

$$\mathrm{den}(A) \neq er \implies \mathrm{den}(P) = 1$$

(Frege, according to Soames)

- $\boxed{\text{Mary has exactly one husband}}$ is a presupposition for both

$$\text{husband(Mary) and tall(man)(husband(Mary))}$$

- Basic examples (in Frege) are descriptions, but also, e.g.,
- Activity verbs: $\mathrm{stop} : \tilde{\mathrm{e}} \times (\tilde{\mathrm{e}} \to \tilde{\mathrm{t}}) \to \tilde{\mathrm{t}}$

$$\mathrm{den}(\mathrm{stopped}(\mathrm{George}, \mathrm{running}))(a)$$

$$= \begin{cases} 1, & \text{if George was running and has stopped in state } a, \\ 0, & \text{if George was running and has not stopped in state } a, \\ er & \text{if George was not running before state } a \end{cases}$$

# Relative clauses

Mary, who loves him, suffers

We assume a constant who : $\tilde{\mathrm{e}} \times (\tilde{\mathrm{e}} \to \tilde{\mathrm{t}}) \to \tilde{\mathrm{e}}$ such that

$$\mathrm{who}(u)(x)(a) = \begin{cases} u(a), & \text{if } x(u,a), \\ er, & \text{otherwise} \end{cases}$$

$$\boxed{\text{Mary, who loves him} \xrightarrow{\text{render}} M \equiv \mathrm{who}(\mathrm{Mary}, \lambda(v)\mathrm{loves}(v, \mathrm{him}))}$$

$$\mathrm{den}(M)(a) = \begin{cases} \mathrm{Mary}(a), & \text{if Mary loves him in state } a, \\ er, & \text{otherwise} \end{cases}$$

$$\boxed{\text{Mary, who loves him, suffers} \xrightarrow{\text{render}} \mathrm{suffers}(M)}$$

$$\mathrm{den}(\mathrm{suffers}(M)(a))$$

$$= \begin{cases} 1, & \text{if Mary loves him and suffers in state } a, \\ 0, & \text{if Mary loves him and does not suffer in state } a, \\ er, & \text{otherwise (if Mary does not love him in state } a) \end{cases}$$

# Computation of errors

- We define $er_{\tilde{\sigma}}$ for every natural language type $\tilde{\sigma}$ by the recursion

$$er_{\tilde{\mathrm{e}}} = er_{\tilde{\mathrm{t}}} = er \quad \text{(given)}$$

$$er_{(\sigma \to \tau)} = h, \text{ where for each } x : \sigma, h(x) = er_\tau$$

- $\boxed{\text{if } A : \tilde{\mathrm{t}} \text{ and } \mathrm{den}(A)(a) = er, \text{ then } \mathrm{den}(\mathrm{not}(A))(a) = er}$

  which is a basic condition for logical presupposition

Basic error propagation rule: *if the computation of* $\mathrm{den}(A)(\mathrm{g})$ *requires* $\mathrm{den}(B)(\mathrm{g}')$ *and* $\mathrm{den}(B)(\mathrm{g}') = er$*, then* $\mathrm{den}(A)(\mathrm{g}) = er$.

- So, if $y(a) = er$, then $\overline{tall}(x, y, a) = er$,
- We may want to allow many error values, which code the (one or many) sources of the problem in computing $\mathrm{den}(A)(\mathrm{g})$.

  In programming languages these are called error messages

# Rigidity

A state dependent object $x : \mathsf{s} \to \sigma$ is rigid if

$$\text{for all states } a, b, x(a) = x(b)$$

Historical proper names are typically assumed to be rigid,

$$\text{Scott}, \quad \text{Aristotle}, \ldots$$

The object

$$\boxed{\mathrm{dere}_\sigma(x, a)(b) = x(a) \quad (x \; : \mathsf{s} \to \sigma, a, b \in \mathbb{T}_\mathsf{s}) : \tilde{\mathsf{e}}}$$

is rigid and denotes $x(a)$ in every state $b$

- There is no obvious English word for the function "dere" (and $\mathrm{dere} : \tilde{\mathsf{e}} \times \mathsf{s} \to \tilde{\mathsf{e}}$ is not of natural language type)

# Modal operators: de dicto and de re interpretations

$$\Box : (\tilde{t} \to \tilde{t}), \qquad \Box(p)(a) \iff (\forall b)p(b) \quad (\text{necessarily always})$$

Consider the following sentence, uttered by Barack Obama in 2010:

$$A \equiv \text{it is necessary that I am American}$$

The rendering $\boxed{A \xrightarrow{\text{render}} \Box(\text{American}(\text{I}))}$ is clearly wrong

$$\Box_1 : \tilde{t} \times \tilde{e} \to \tilde{t}, \quad \Box_1(p, x)(a) = \Box(p(\text{dere}(x, a)))(a), \text{ so that}$$

$$\boxed{\mathfrak{M}_0 \models \Box_1(p, x)(a) \iff x(a) \text{ necessarily has property } p}$$

$$\boxed{\text{It is necessary that I am American} \xrightarrow{\text{render}} \Box_1(\text{American}, \text{I})}$$

which (uttered by Obama) says that *Obama is necessarily American*

- *Kaplan's interpretation of modal sentences with demonstratives*

# Local and modal dependence

- $p : (\mathsf{s} \to \sigma_1) \times (\mathsf{s} \to \sigma_1) \to (\mathsf{s} \to \tau)$ is
  — local in the first argument if $p(x, y)(a) = p(\mathrm{dere}(x, a), y)(a)$
  — local in the second argument if $p(x, y)(a) = p(x, \mathrm{dere}(y, a))(a)$
  otherwise $p$ is modal in the relevant argument
- $\Box$ is modal
  $\Box_1$ is modal in its first argument, local in the second
- and, runs, loves, etc. are local in all their arguments

$$\text{the temperature is rising} \xrightarrow{\text{render}} \text{rises}(\text{the}(\text{temp})), \quad \text{where}$$

$$\text{temp}(x)(a) \iff \text{the temperature in state } a \text{ is } x(a) \text{ degrees (local)},$$

- rises is a modal verb (Partee)

$$a\{j := t\} = \text{the state which is } a \text{ except that time}(a\{j := t\}) = t,$$

$$\text{rises}(x, a) \iff \text{the function } t \mapsto x(a\{j := t\}) \text{ is increasing at time}(a)$$

$$\iff \frac{\partial x(a\{j := t\})}{\partial t}(a) > 0$$

# Coindexing in the $\lambda$-calculus

$$\begin{aligned}
\text{John loves himself} &\xrightarrow{\text{formalize}} \text{loves}(\text{John}, \text{himself})\\
&\xrightarrow{\text{coindex}}_{\lambda} \Big(\lambda(\jmath)\text{loves}(\jmath, \jmath)\Big)(\text{John})\\
\text{John kissed his wife} &\xrightarrow{\text{formalize}} \text{kissed}(\text{John}, \text{wife}(\text{his}))\\
&\xrightarrow{\text{coindex}}_{\lambda} \Big(\lambda(\jmath)\text{kissed}(\jmath, \text{wife}(\jmath))\Big)(\text{John})
\end{aligned}$$

John loves his wife and he honors her

$$\begin{aligned}
&\xrightarrow{\text{formalize}} \text{loves}(\text{John}, \text{wife}(\text{his}))\ \&\ \text{honors}(\text{he}, \text{her})\\
&\xrightarrow{\text{coindex}}_{\lambda} \lambda(\jmath)\Big[\text{loves}(\jmath, \text{wife}(\jmath))\ \&\ \text{honors}(\jmath, \text{her})\Big](\text{John})\\
&\xrightarrow{\text{coindex}}_{\lambda} \lambda(\jmath)\Big[\lambda(w)\Big(\text{loves}(\jmath, w)\ \&\ \text{honors}(\jmath, w)\Big)(\text{wife}(\jmath))\Big](\text{John})
\end{aligned}$$

- $\xrightarrow{\text{render}} \quad = \quad \xrightarrow{\text{formalize}} + \xrightarrow{\text{coindex}}_1 + \cdots + \xrightarrow{\text{coindex}}_k$
- *The last $\lambda$-rendering turns a conjunction into a predication*

# Coindexing in $\mathsf{L}^{\lambda}_{\mathrm{ar}}$

$$\text{John loves himself} \xrightarrow{\text{formalize}} \text{loves}(\text{John}, \text{himself})$$

$$\xrightarrow{\text{coindex}}_{\mathrm{ar}} \text{loves}(j, j) \text{ where } \{j := \text{John}\}$$

$$\text{John kissed his wife} \xrightarrow{\text{formalize}} \text{kissed}(\text{John}, \text{wife}(\text{his}))$$

$$\xrightarrow{\text{coindex}}_{\mathrm{ar}} \text{kissed}(j, \text{wife}(j)) \text{ where } \{j := \text{John}\}$$

John loves his wife and he honors her

$$\xrightarrow{\text{formalize}} \text{loves}(\text{John}, \text{wife}(\text{his})) \,\&\, \text{honors}(\text{he}, \text{her})$$

$$\xrightarrow{\text{coindex}}_{\mathrm{ar}} \text{loves}(j, \text{wife}(j)) \,\&\, \text{honors}(j, \text{her}) \text{ where } \{j := \text{John}\}$$

$$\xrightarrow{\text{coindex}}_{\mathrm{ar}} \Big(\text{loves}(j, \dot{w}) \,\&\, \text{honors}(j, \dot{w}) \text{ where } \{\dot{w} := \text{wife}(j)\}\Big)$$

$$\text{where } \{j := \text{John}\}$$

$$\approx_{\ell} \text{loves}(j, \dot{w}) \,\&\, \text{honors}(j, \dot{w}) \text{ where } \{\dot{w} := \text{wife}(j), j := \text{John}\}$$

- *The last* $\mathsf{L}^{\lambda}_{\mathrm{ar}}$*-rendering produces a conjunction*

# Renderings which involve coindexing in $\text{Ty}_2$ and $\text{L}^{\lambda}_{\text{ar}}$

$$\text{John loves himself} \xrightarrow{\text{render}}_{\lambda} \Big(\lambda(\jmath)\text{loves}(\jmath,\jmath)\Big)(\text{John}) \tag{1a}$$

$$\text{John loves himself} \xrightarrow{\text{render}} \text{loves}(j,j) \text{ where } \{j := \text{John}\} \tag{1b}$$

$$\text{John kissed his wife} \xrightarrow{\text{render}}_{\lambda} \Big(\lambda(\jmath)\text{kissed}(\jmath,\text{wife}(\jmath))\Big)(\text{John}) \tag{2a}$$

$$\text{John kissed his wife} \tag{2b}$$

$$\xrightarrow{\text{render}} \text{kissed}(j,\text{wife}(j)) \text{ where } \{j := \text{John}\}$$

$$\text{John loves his wife and he honors her} \tag{3a}$$

$$\xrightarrow{\text{render}}_{\lambda} \lambda(\jmath)\Big[\lambda(w)\Big(\text{loves}(\jmath,w)\,\&\,\text{honors}(\jmath,w)\Big)(\text{wife}(\jmath))\Big](\text{John})$$

$$\text{John loves his wife and he honors her} \tag{3b}$$

$$\xrightarrow{\text{render}} \text{loves}(j,\dot{w})\,\&\,\text{honors}(j,\dot{w}) \text{ where } \{\dot{w} := \text{wife}(j), j := \text{John}\}$$

(1a) $\approx_{\ell}$ (1b) $\boxed{\text{(2b) and (3b) are not synonymous with any } \text{Ty}_2 \text{ terms}}$

# Proper nouns, demonstratives and quantifiers

Montague renders proper names by their evaluation quantifier:

$$\mathsf{John}_{\mathsf{Mont}}(p) = p(\mathsf{John})$$

so we get similar renderings for predication and quantification

$$\text{John runs} \xrightarrow{\text{render}} \mathsf{John}_{\mathsf{Mont}}(\mathsf{runs}),\ \text{every man runs} \xrightarrow{\text{render}} \mathsf{every}(\mathsf{man})(\mathsf{runs})$$

We follow the simpler type driven rendering by which John : $\tilde{\mathsf{e}}$,

$$\text{John runs} \xrightarrow{\text{render}} \mathsf{runs}(\mathsf{John}),\ \text{every man runs} \xrightarrow{\text{render}} \mathsf{every}(\mathsf{man})(\mathsf{runs})$$

$$\begin{aligned}\text{John loves every girl} &\xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{every}(\mathsf{girl}))\\ &\xrightarrow{\text{render}} \mathsf{every}(\mathsf{girl})(\lambda(u)\mathsf{loves}(\mathsf{John}, u))\end{aligned}$$

# Coordination using acyclic recursion

John entered the room and Mary entered the room (conjunction)

John and Mary entered the room (predication)

John and Mary entered the room

$$\xrightarrow{\text{render}}_{\lambda} \lambda(r)\Big(r(\text{John}) \text{ and } r(\text{Mary})\Big)(\text{entered})$$

John and Mary entered the room

$$\xrightarrow{\text{render}} \lambda(r)\Big(r(j)) \text{ and } r(\dot{m})\Big)(\text{entered}) \text{ where } \{j := \text{John}, \dot{m} := \text{Mary}\}$$

- *These two renderings are not synonymous* (Perhaps they should be!)

The teacher and every student laughed

$$\xrightarrow{\text{render}} \lambda(r)\Big(r(\dot{t}) \text{ and } \dot{s}(r)\Big)(\text{laughed})$$

$$\text{where } \{\dot{t} := \text{the}(\text{teacher}), \dot{s} := \text{every}(\text{student})\}$$

# Identity statements

Frege's original puzzle was about the identity statement

$$\text{the morning star} = \text{the evening star}$$

Montague would render this and its converse by

The evening star is the morning star

$$\xrightarrow{\text{render}} \mathrm{ES}_{\mathrm{Mont}}(\lambda(u)\mathrm{MS}_{\mathrm{Mont}}(\lambda(v)(u = v))),$$

The morning star is the evening star

$$\xrightarrow{\text{render}} \mathrm{MS}_{\mathrm{Mont}}(\lambda(u)\mathrm{ES}_{\mathrm{Mont}}(\lambda(v)(u = v)))$$

- *These two terms are not referentially synonymous*

With the type-driven renderings we use, for any $A, B : \sigma$

$$A = B \xrightarrow{\text{render}} =_\sigma (A, B), \quad B = A \xrightarrow{\text{render}} =_\sigma (B, A)$$
$$=_\sigma (A, B) \approx_\ell =_\sigma (B, A)$$

- *We understand these terms as expressing identity statements*

(as Frege intended them to be understood)

# Overview: The Reduction Calculus

(1) We will define a reduction relation between terms so that intuitively

$$\boxed{\begin{aligned} A \Rightarrow B \iff\ & A \equiv_c B \quad (A \text{ is congruent with } B) \\ & \text{or } A \text{ and } B \text{ have the same meaning} \\ & \quad \text{and } B \text{ expresses that meaning "more directly"} \end{aligned}}$$

- (Some terms, e.g., variables, will not be assigned meanings)
- $A \Rightarrow A$, $(A \Rightarrow B \text{ and } B \Rightarrow C) \implies A \Rightarrow C$
- Compositionality: $C_1 \Rightarrow C_2 \implies A\{x :\equiv C_1\} \Rightarrow A\{x :\equiv C_2\}$
- $A \Rightarrow B$ is defined by ten simple rules, like a proof system

A term $A$ is irreducible if

$$A \Rightarrow B \implies A \equiv_c B.$$

- Meaningful irreducible terms express their meaning directly: their meaning is exhausted by their denotation

# Overview: Canonical and Logical Forms

## Canonical Form Theorem

*For each term $A$, there is a recursive, irreducible term*

$$\boxed{\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}}$$

*such that each $A_i$ is explicit, irreducible and* $\boxed{A \Rightarrow \mathrm{cf}(A)}$

*Moreover,* cf($A$) *can be effectively computed and is the unique* (up to congruence) *irreducible term to which $A$ can be reduced, i.e.,*

$$\textit{if } A \Rightarrow B \textit{ and } B \textit{ is irreducible, then } B \equiv_c \mathrm{cf}(A)$$

We write: $\boxed{A \Rightarrow_{\mathrm{cf}} B \iff \mathrm{cf}(A) \equiv_c B}$

$A_0, A_1, \ldots, A_n$ *are the parts of $A$, $n$ is its dimension*

- $\boxed{\mathrm{cf}(A) \textit{ models the logical form of } A}$

# Overview: Referential intensions and truth conditions

- Variables and some simple immediate terms have no meaning, they refer immediately.
- Constants, refer directly, but they have meanings, albeit trivial ones which are exhausted by their denotations

*The distinction between immediate and direct reference is a central feature of referential intension theory*

- If $A$ is *proper* (not immediate) and

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\},$$

  then the referential intension $\mathrm{int}(A)$ of $A$ is (intuitively) the abstract algorithm which computes for each valuation g the denotation $\mathrm{den}(A)(\mathrm{g})$ as in the examples above: we solve the acyclic system to find $\overline{p}_1, \ldots, \overline{p}_n$ and set

$$\mathrm{den}(A)(\mathrm{g}) = \mathrm{den}(A_0)(\mathrm{g}\{\dot{p}_1 := \overline{p}_1, \ldots, \dot{p}_n := \overline{p}_n\})$$

- The parts $A_0, \ldots, A_n$ are the (generalized) truth conditions for $A$

# Overview: Referential and Logical Synonymy

## Referential Synonymy Theorem

*Two proper terms $A, B$ are referentially synonymous if and only if*

$$A \Rightarrow_{\mathrm{cf}} A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}$$
$$B \Rightarrow_{\mathrm{cf}} B_0 \text{ where } \{\dot{p}_1 := B_1, \ldots, \dot{p}_n := B_n\}$$

*for some $A_0, A_1, \ldots, A_n, B_0, B_1, \ldots, B_n$ such that*

(RS1) *The corresponding parts $A_i, B_i$ of $A$ and $B$ have the same free variables, i.e., for every variable $x$ of either kind,*

$$x \text{ occurs free in } A_i \iff x \text{ occurs free in } B_i, \quad (i = 0, \ldots, n).$$

(RS2) $\mathfrak{M}_0 \models A_i = B_i, \quad (i = 0, 1, \ldots, n)$

Def. $A \approx_\ell B \iff A$ is logically synonymous with $B$

$\iff$ (RS1) and (RS2$_l$) : $\models A_i = B_i, \quad (i = 0, 1, \ldots, n)$

C. L. Dodgson $\approx$ Lewis Carroll but C. L. Dodgson $\not\approx_\ell$ Lewis Carroll

# Overview: The calculi of referential and logical synonymy

$\boxed{\sim \text{ is either synonymy } \approx \textbf{ or } \text{logical synonymy } \approx_\ell}$

$$\frac{A \Rightarrow B}{A \sim B} \qquad A \sim A \qquad \frac{A \sim B}{B \sim A} \qquad \frac{A \sim B \qquad B \sim C}{A \sim C}$$

$$\frac{A_1 \sim B_1 \qquad A_2 \sim B_2}{A_1(A_2) \sim B_1(B_2)} \qquad \frac{A \sim B}{\lambda(u)A \sim \lambda(u)B}$$

$$\frac{A_0 \sim B_0, \qquad A_1 \sim B_1, \qquad \ldots, \qquad A_n \sim B_n}{A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots \dot{p}_n := A_n\} \sim B_0 \text{ where } \{\dot{p}_1 := B_1, \ldots, \dot{p}_n := B_n\}}$$

If $C, D$ are immediate **or** proper, explicit irreducible, $\mathrm{FV}(C) = \mathrm{FV}(D)$

$$\frac{\mathfrak{M}_0 \models C = D}{C \approx D} \qquad \frac{\models C = D}{C \approx_\ell D}$$

- The proof systems are complete. The framed rule is not effective for $\approx$ because $\underline{\mathfrak{M}_0 \models C = D \text{ is not decidable}}$ (?)

# The Reduction Calculus: the basic rules

Congruence, Transitivity, Compositionality

(cong) If $A \equiv_c B$, then $A \Rightarrow B$

(trans) If $A \Rightarrow B$ and $B \Rightarrow C$, then $A \Rightarrow C$

(rep1) If $A \Rightarrow A'$ and $B \Rightarrow B'$, then $A(B) \Rightarrow A'(B')$

(rep2) If $A \Rightarrow B$, then $\lambda(u)(A) \Rightarrow \lambda(u)(B)$

(rep3) If $A_i \Rightarrow B_i$ for $i = 0, \ldots, n$, then

$$A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}$$
$$\Rightarrow B_0 \text{ where } \{\dot{p}_1 := B_1, \ldots, \dot{p}_n := B_n\}$$

- These do not produce any non-trivial reductions

# The Reduction Calculus: rules for recursion

For distinct locations $\dot{p}_1, \ldots, \dot{p}_n$ and $\dot{q}_1, \ldots, \dot{q}_m$, and terms $A_i, B_j$, set

$$\vec{\dot{p}} := \vec{A} \ \text{ for } \ \dot{p}_1 := A_1, \ldots, \dot{p}_n = A_n,$$
$$\vec{\dot{q}} := \vec{B} \ \text{ for } \ \dot{q}_1 := B_1, \ldots, \dot{q}_m = B_m,$$

(head) $\Big(A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}\}\Big) \text{ where } \{\vec{\dot{q}} := \vec{B}\}$

$$\Rightarrow A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}, \vec{\dot{q}} := \vec{B}\}$$

(B-S) $A_0 \text{ where } \{\dot{r} := \Big(B_0 \text{ where } \{\vec{\dot{q}} := \vec{B}\}\Big), \vec{\dot{p}} := \vec{A}\}$

$$\Rightarrow A_0 \text{ where } \{\dot{r} := B_0, \vec{\dot{q}} := \vec{B}, \vec{\dot{p}} := \vec{A}\}$$

- These just allow the "parallel" combination of assignments

(recap) $\boxed{\Big(A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}\}\Big)(B) \Rightarrow A_0(B) \text{ where } \{\vec{\dot{p}} := \vec{A}\}}$

- The (recap) rule has an important consequence for any notion of meaning which is preserved by reduction

# The Reduction Calculus: import of the (recap) rule

- We will see (after the next rule) that

$$\text{I am tall} \xrightarrow{\text{render}} \text{tall}(\text{I}) \Rightarrow_{\text{cf}} \boxed{A \equiv \text{tall}(i) \text{ where } \{i := \text{I}\}}$$

- Let $u$ be a state variable and $\mathrm{g}(u) = a$: clearly

$$\text{den}(A(u))(\mathrm{g}) = 1 \iff \text{the speaker in state } a \text{ is tall in state } a$$

and so to compute $\text{den}(A(u))(\mathrm{g})$ we only need to know the property of "tallness" in state $a$ and speaker$(a)$

- By the (recap) rule $\boxed{A(u) \Rightarrow \text{tall}(i)(u) \text{ where } \{i := \text{I}\}}$
- To compute $\text{den}(A(u))(\mathrm{g})$ using this expression we need to know what "tall" means in $a$ and the entire meaning of "I"—that it assigns to every state $b$ the entity speaker$(b)$

*The meaning of* $\Big(A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}\Big)(B)$
*depends on the meanings of* $A_0(B), A_1, \ldots, A_n$

# The Reduction Calculus: immediate terms

- Immediate applicative terms

$$E :\equiv u \mid \dot{p} \mid x(v_1, \dots, v_n) \mid \dot{p}(v_1, \dots, v_n)$$

  where $x, u, v_1, \dots, v_n$ are pure variables and $\dot{p}$ is a recursion variable (and the types match)
- Immediate $\lambda$-terms $X :\equiv \lambda(v_1, \dots, v_n)(E)$
- $X$ is immediate if it is applicative or $\lambda$-immediate
  $A$ is proper if it is not immediate
- Immediate terms act like generalized variables:
  they denote immediately and they cannot be assigned meanings
- Plausible: $\boxed{\textit{Natural language phrases are rendered by proper terms}}$
  A strong version of the view that

  $\boxed{\text{there are no true variables in natural language}}$

# The Reduction Calculus: the application rule

$$\boxed{\text{(ap)} \quad A(B) \Rightarrow A(\dot{b}) \text{ where } \{\dot{b} := B\} \quad (B \text{ proper}, \ \dot{b} \text{ fresh})}$$

- John is tall $\xrightarrow{\text{render}}$ tall(John) $\Rightarrow_{\mathrm{cf}}$ tall($j$) where $\{j := \text{John}\}$

- Peter likes the blond $\xrightarrow{\text{render}}$ likes(Peter)(the(blond))

$$\begin{aligned}
&\Rightarrow \text{likes(Peter)}(\dot{b}) \text{ where } \{\dot{b} = \text{the(blond)}\} && \text{(ap)}\\
&\Rightarrow \text{likes(Peter)}(\dot{b})\\
&\quad\ \text{where } \{\dot{b} = \text{the}(\dot{B}) \text{ where } \{\dot{B} := \text{blond}\}\} && \text{(ap, rep)}\\
&\Rightarrow \text{likes(Peter)}(\dot{b}) \text{ where } \{\dot{b} = \text{the}(\dot{B}), \dot{B} := \text{blond}\} && \text{(B-S)}\\
&\Rightarrow \Big(\text{likes}(\dot{p}) \text{ where } \{\dot{p} := \text{Peter}\}\Big)(\dot{b})\\
&\qquad\qquad \text{where } \{\dot{b} = \text{the}(\dot{B}), \dot{B} := \text{blond}\} && \text{(ap,rep)}\\
&\Rightarrow \Big(\text{likes}(\dot{p})(\dot{b}) \text{ where } \{\dot{p} := \text{Peter}\}\Big)\\
&\qquad\qquad \text{where } \{\dot{b} = \text{the}(\dot{B}), \dot{B} := \text{blond}\} && \text{(recap,rep)}
\end{aligned}$$

$$\Rightarrow \boxed{\text{likes}(\dot{p})(\dot{b}) \text{ where } \{\dot{p} := \text{Peter}, \dot{b} = \text{the}(\dot{B}), \dot{B} := \text{blond}\}}$$

the last by (head,S-B,rep)

# The Reduction Calculus: an example of the $\lambda$-rule

every man danced with his wife

$$\xrightarrow{\text{render}} A \equiv \text{every}(\text{man})\Big(\lambda(u)\text{danced}(u, \text{wife}(u))\Big)$$

$$\Rightarrow \text{every}(\text{man})\Big[\lambda(u)\Big(\text{danced}(u, \dot{w}) \text{ where } \{\dot{w} := \text{wife}(u)\}\Big)\Big] \quad \text{(ap,rep)}$$

- $A$ says that "every man has property $R$", where, for each $u$,

$$R(u) \iff u \text{ danced with wife}(u)$$

So we want

$$\lambda(u)\Big(\text{danced}(u, \dot{w}) \text{ where } \{\dot{w} := \text{wife}(u)\}\Big)$$
$$\Rightarrow \lambda(u)\text{danced}(u, \dot{w}'(u)) \text{ where } \{\dot{w}' := \lambda(u)\text{wife}(u)\}$$

- The rule "distributes the $\lambda$" over the parts of its scope

# The Reduction Calculus: the $\lambda$-rule

$$\text{($\lambda$-rule)} \quad \lambda(u)\Big(A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}\Big)$$
$$\Rightarrow \lambda(u)A_0' \text{ where } \{\dot{p}_1' := \lambda(u)A_1', \ldots, \dot{p}_n' := \lambda(u)A_n'\}$$

where for $i = 1, \ldots, n$, $\dot{p}_i'$ is a fresh recursion variable and

$$A_i' :\equiv A_i\{\dot{p}_1 :\equiv \dot{p}_1'(u), \ldots, \dot{p}_n :\equiv \dot{p}_n'(u)\}.$$

# The Reduction Calculus

(cong) If $A \equiv_c B$, then $A \Rightarrow B$

(trans) If $A \Rightarrow B$ and $B \Rightarrow C$, then $A \Rightarrow C$

(rep1) If $A \Rightarrow A'$ and $B \Rightarrow B'$, then $A(B) \Rightarrow A'(B')$

(rep2) If $A \Rightarrow B$, then $\lambda(u)(A) \Rightarrow \lambda(u)(B)$

(rep3) If $A_i \Rightarrow B_i$ for $i = 0, \dots, n$, then

$$A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}\} \Rightarrow B_0 \text{ where } \{\vec{\dot{p}} := \vec{B}\}$$

(head) $\Big(A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}\}\Big) \text{ where } \{\vec{\dot{q}} := \vec{B}\} \Rightarrow A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}, \vec{\dot{q}} := \vec{B}\}$

(B-S) $A_0 \text{ where } \{\dot{r} := \Big(B_0 \text{ where } \{\vec{\dot{q}} := \vec{B}\}\Big), \vec{\dot{p}} := \vec{A}\}$

$$\Rightarrow A_0 \text{ where } \{\dot{r} := B_0, \vec{\dot{q}} := \vec{B}, \vec{\dot{p}} := \vec{A}\}$$

---

(recap) $\Big(A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}\}\Big)(B) \Rightarrow A_0(B) \text{ where } \{\vec{\dot{p}} := \vec{A}\}$

(ap) $A(B) \Rightarrow A(\dot{b}) \text{ where } \{\dot{b} := B\}$ ($B$ proper, $\dot{b}$ fresh)

($\lambda$-rule) $\lambda(u)(A_0 \text{ where } \{\vec{\dot{p}} := \vec{A}\}) \Rightarrow \lambda(u)A'_0 \text{ where } \{\overrightarrow{\dot{p}'} := \overrightarrow{\lambda(u)A'}\}$

# Two simple results

**Theorem** (Reduction preserves denotations)

*If* $A \Rightarrow B$, *then* $\models A = B$

Proof is by induction on $\Rightarrow$ (simple). $\square$

**Theorem** (Characterization of irreducible terms)

(a) *Constants and immediate terms are irreducible.*
(b) *An application term* $A(B)$ *is irreducible if and only if* $B$ *is immediate and* $A$ *is explicit and irreducible.*
(c) *A* $\lambda$*-term* $\lambda(u)(A)$ *is irreducible if and only if* $A$ *is explicit and irreducible.*
(d) *A recursive term* $A_0$ where $\{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}$ *is irreducible if and only all the parts* $A_0, \ldots, A_n$ *are explicit and irreducible.*

Proof is by inspection of the reduction rules (simple). $\square$

# Canonical forms

For each term $A$, we define by recursion $\mathrm{cf}(A)$ so that:

**Theorem**

(1) $\boxed{\mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \qquad (n \geq 0)}$

*with explicit, irreducible parts $A_0, A_1, \ldots, A_n$, so that it is irreducible*

*A constant $c$ or a variable $x$ occurs free in $\mathrm{cf}(A)$ if and only if it occurs free in $A$*

(2) $A \Rightarrow \mathrm{cf}(A)$

(3) *If $A$ is irreducible, then $\mathrm{cf}(A) \equiv A$*

(4) $\boxed{\text{If } A \Rightarrow B, \text{ then } \mathrm{cf}(A) \equiv_c \mathrm{cf}(B)}$

(5) *If $A \Rightarrow B$ and $B$ is irreducible, then $B \equiv_c \mathrm{cf}(A)$*

- (1) and (2) are easy, (3) is trivial and (5) follows from (3) and (4)
- The proof of (4) is by induction on $A \Rightarrow B$; complex

# Every man danced with his wife (if wife is a constant)

every man danced with his wife

$$\xrightarrow{\text{render}} A \equiv \text{every}(\text{man})\Big(\lambda(u)\Big(\text{danced}(u, \dot{w}) \text{ where } \{\dot{w} := \text{wife}(u)\}\Big)\Big)$$

$$\Rightarrow \text{every}(\text{man})\Big(\lambda(u)\text{danced}(u, \dot{w}'(u))$$

$$\text{where } \{\dot{w}' := \lambda(u)\text{wife}(u)\}\Big)$$

$$\Rightarrow \text{every}(\dot{m}, \dot{d}) \text{ where } \Big\{\dot{m} := \text{man},$$

$$\dot{d} := \lambda(u)\text{danced}(u, \dot{w}(u)) \text{ where } \{\dot{w} := \lambda(u)\text{wife}(u)\}\Big\}$$

$$\Rightarrow_{\text{cf}} \text{every}(\dot{m}, \dot{d}) \text{ where}$$

$$\{\dot{m} := \text{man}, \dot{d} := \lambda(u)\text{danced}(u, \dot{w}(u)), \dot{w} := \lambda(u)\text{wife}(u)\}$$

$$\approx_\ell \text{every}(\dot{m}, \dot{d}) \text{ where } \{\dot{m} := \text{man}, \dot{d} := \lambda(u)\text{danced}(u, \dot{w}(u)), \dot{w} := \text{wife}\}$$

- *The reduction is more complex if wife(u)* $\equiv$ *the*$(\lambda(v)$*married*$(u, v))$

# Shapes $\boxed{\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}}$

$$\begin{aligned}\vec{\mathbf{f}}_i(A) &= \text{a listing of the variables which are free in } A_i \text{ and } A \\ &= \mathrm{FV}(A_i) \cap \mathrm{FV}(A) \quad (i = 0, \dots, n) \\ \vec{\mathbf{r}}_i(A) &= \text{a listing of the locations } (\dot{p}_1, \dots, \dot{p}_n) \text{ which occur free in } A_i \\ &= \mathrm{FV}(A_i) \cap (\dot{p}_1, \dots, \dot{p}_n) \quad (i = 0, \dots, n)\end{aligned}$$

$$\boxed{\mathrm{shape}(A) = (\dot{p}_1, \dots, \dot{p}_n, \vec{\mathbf{f}}_0(A), \vec{\mathbf{r}}_0(A), \dots, \vec{\mathbf{f}}_n(A), \vec{\mathbf{r}}_n(A))}$$

$$\mathrm{shape}\Big(\mathrm{every}(\dot{m})(\dot{l}) \text{ where } \{\dot{m} := \mathrm{man}, \dot{l} := \lambda(x)\mathrm{loves}(x, e)\}\Big)$$
$$= (\dot{m}, \dot{l}, \underline{\emptyset, \langle \dot{m}, \dot{l}\rangle}, \underline{\emptyset, \emptyset}, \underline{\langle e\rangle, \emptyset})$$

- shape($A$) codes the number of parts of $A$, the free variables of each part, and the putative dependence relation

$$\dot{p}_i \rightarrow \dot{p}_j \iff \dot{p}_j \text{ occurs free in } \dot{A}_i$$

- Synonymous terms have isomorphic shapes

# Referential intensions

$\boxed{\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}}$

- $\alpha_i(\mathrm{g}, d_1, \ldots, d_n) = \mathrm{den}(A_i)(\mathrm{g}\{\dot{p}_1 := d_1, \ldots, \dot{p}_n := d_n\}) \quad (i \leq n)$
- $\mathrm{system}(A) = (\alpha_0, \ldots, \alpha_n)$
- $\boxed{\mathrm{int}(A) = (\mathrm{shape}(A), \mathrm{system}(A))}$

For example,

$$\begin{aligned}
&\mathrm{int}(\mathrm{loves}(\mathrm{Abelard}, \mathrm{Eloise})) \\
&\qquad = \mathrm{int}(\mathrm{loves}(\dot{a}, \dot{e}) \text{ where } \{\dot{a} := \mathrm{Abelard}, \dot{e} = \mathrm{Eloise}\}) \\
&\qquad\qquad = ((\dot{a}, \dot{e}, \emptyset, \langle \dot{a}, \dot{e} \rangle, \emptyset, \emptyset, \emptyset, \emptyset), (\alpha_0, \alpha_1, \alpha_2)),
\end{aligned}$$

$$\begin{aligned}
\text{where } \alpha_0(\mathrm{g}, d_1, d_2) &= \mathrm{den}(\mathrm{loves}(\dot{a}, \dot{e}))(\mathrm{g}\{\dot{a} := d_1, \dot{e} := d_2\}) \\
&= \mathrm{loves}(d_1, d_2) \\
\alpha_1(\mathrm{g}) = \mathrm{Abelard}, &\quad \alpha_2(\mathrm{g}) = \mathrm{Eloise}
\end{aligned}$$

- $\mathrm{int}(A)$ is an acyclic recursor
- There is a simple notion of natural isomorphism $\cong$ between acyclic recursors, and $\mathrm{int}(A)$ is defined up to natural isomorphism

# What is an algorithm?

The referential intension of a term $A$ is an acyclic recursor

$$\boxed{\mathrm{int}(A) = (\mathrm{shape}(A), \alpha_0, \ldots, \alpha_n)}$$

where $\alpha_0, \ldots, \alpha_n$ are functions (on valuations and objects of $\mathfrak{M}_0$) and shape($A$) codes some (basically syntactic) information about the variables of $\alpha_i$ and their "interdependence"

In what sense is this an algorithm?

- There is no general agreement on what algorithms are
- There is a robust class of computable functions associated with each *first order structure*, characterized by various models of computation—Turing machines, etc.
  This is the Church-Turing Thesis
- The recursive programs of McCarthy is (perhaps) the most natural model of computation; and its direct generalization to structures $\mathfrak{M}$ for $\mathrm{L}^{\lambda}_{\mathrm{ar}}(K)$ yields a class of algorithms which includes the acyclic recursors we use to model meanings

# Referential and logical synonymy

$$A \approx B \iff A, B \text{ are immediate and } \mathfrak{M}_0 \models A = B,$$
$$\text{or } A \text{ and } B \text{ are proper and } \mathrm{int}(A) \cong \mathrm{int}(B)$$

**Referential Synonymy Theorem**

*Two proper terms $A, B$ are referentially synonymous if and only if*

$$A \Rightarrow_{\mathrm{cf}} A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}$$
$$B \Rightarrow_{\mathrm{cf}} B_0 \text{ where } \{\dot{p}_1 := B_1, \ldots, \dot{p}_n := B_n\}$$

*for some $A_0, A_1, \ldots, A_n, B_0, B_1, \ldots, B_n$ such that*

(RS1) *The corresponding parts $A_i, B_i$ of $A$ and $B$ have the same free variables, i.e., for every variable $x$ of either kind,*

$$x \text{ occurs free in } A_i \iff x \text{ occurs free in } B_i, \quad (i = 0, \ldots, n).$$

(RS2) $\mathfrak{M}_0 \models A_i = B_i, \quad (i = 0, 1, \ldots, n)$

or (RS2′) $\models A_i = B_i, \quad (i = 0, 1, \ldots, n)$ *for logical synonymy*

# Why not assign meanings to immediate terms?

Suppose there is a constant id : (e $\to$ e) for the identity function

$$\overline{\mathrm{id}}(x) = x \quad (x : \mathrm{e})$$

Now both $\mathrm{id}(x)$ and $x$ are irreducible with the same denotation, and so *they would be synonymous if $x$ had a referential intension*,

$$\mathrm{id}(x) \approx x$$

Then for any constant c,

$$\mathrm{c}(\mathrm{id}(x)) \Rightarrow_{\mathrm{cf}} \mathrm{c}(\dot{p}) \text{ where } \{\dot{p} := \mathrm{id}(x)\}, \quad \mathrm{c}(x) \Rightarrow_{\mathrm{cf}} \mathrm{c}(x) \text{ where } \{\ \}$$

so that

$$\mathrm{id}(x) \approx x \text{ but } \mathrm{c}(\mathrm{id}(x)) \not\approx \mathrm{c}(x),$$

violating compositionality

- *Immediate terms behave like* 0 *in the arithmetic of fractions: it is not possible to assign a value to* $\frac{1}{0}$ *without violating the usual laws of that arithmetic (the field axioms)*

# Using the axioms to prove synonymies

For proper $A, B$

$$\begin{aligned} A = B \ \Rightarrow\ & \dot{a} = \dot{b} \text{ where } \{\dot{a} := A, \dot{b} := B\} \\ \equiv_c\ & \dot{a} = \dot{b} \text{ where } \{\dot{b} := B, \dot{a} := A\} \\ \approx_\ell\ & \dot{b} = \dot{a} \text{ where } \{\dot{b} := B, \dot{a} := A\} \\ \approx_\ell\ & B = A, \end{aligned}$$

where the crucial, second step is valid because

$$\models (\dot{a} = \dot{b} \iff \dot{b} = \dot{a}).$$

- It is more difficult to establish non-synonymy, which (basically) requires the full computation of canonical forms

# The unique occurrence property of explicit terms

Recall: $A$ is explicit if no "where" occurs in it
—so it is a $\mathsf{Ty}_2$ term with (perhaps) some recursive variables in it.

**Theorem**

*No location occurs in more than one part of an explicit term*

**Corollary**

*If a location $\dot{p}$ occurs in two parts $A_k$ and $A_l$ of a term $A$, then $A$ is not referentially synonymous with any explicit term*

The theorem is proved by induction on the definition of explicit terms (using the reduction calculus), and the corollary follows by the Referential Synonymy Theorem

# New meanings expressed in $\mathsf{L}^{\lambda}_{\mathsf{ar}}$

$$\text{John kissed his wife} \xrightarrow{\text{formalize}} \text{kissed}(\text{John}, \text{wife}(\text{his}))$$

$$\xrightarrow{\text{coindex}} \text{kissed}(j, \text{wife}(j)) \text{ where } \{j := \text{John}\}$$

$$\Rightarrow_{\text{cf}} \text{kissed}(j, \dot{w}) \text{ where } \{j := \text{John}, \dot{w} := \text{wife}(j)\}$$

John loves his wife and he honors her

$$\xrightarrow{\text{formalize}} \text{loves}(\text{John}, \text{wife}(\text{his})) \text{ and honors}(\text{he}, \text{her})$$

$$\xrightarrow{\text{render}} \text{loves}(j, \dot{w})\, \&\, \text{honors}(j, \dot{w}) \text{ where } \{\dot{w} := \text{wife}(j), j := \text{John}\}$$

$$\Rightarrow_{\text{cf}} (\dot{l} \text{ and } \dot{h}) \text{ where } \{\dot{l} := \text{loves}(j, \dot{w}), \dot{h} := \text{honors}(j, \dot{w})$$
$$\dot{w} := \text{wife}(j), j := \text{John}\}$$

# New meanings expressed in $L^{\lambda}_{ar}$; logical form

$$A : \boxed{\text{John stumbled and fell}} \xrightarrow{\text{render}} \lambda(u)(\text{stumbled}(u) \text{ and } \text{fell}(u))(\text{John})$$

$$\Rightarrow_{\text{cf}} \Big(\lambda(u)(\dot{s}(u) \text{ and } \dot{f}(u))\Big)(j)$$

$$\text{where } \{\dot{s} := \lambda(u)\text{stumbled}(u), \dot{f} := \lambda(u)\text{fell}(u), j := \text{John}\}$$

$$\approx_{\ell} \Big(\lambda(u)(\dot{s}(u) \text{ and } \dot{f}(u))\Big)(j) \text{ where } \{\dot{s} := \text{stumbled}, \dot{f} := \text{fell}, j := \text{John}\}$$

$$B : \boxed{\text{John stumbled and he fell}} \xrightarrow{\text{formalize}} \text{stumbled}(\text{John}) \text{ and } \text{fell}(\text{John})$$

$$\xrightarrow{\text{coindex}} \text{stumbled}(j) \text{ and } \text{fell}(j) \text{ where } \{j := \text{John}\}$$

$$\Rightarrow_{\text{cf}} (\dot{s} \text{ and } \dot{f}) \text{ where } \{\dot{s} := \text{stumbled}(j), \dot{f} := \text{fell}(j), j := \text{John}\}$$

- Is there a difference in the logical meanings of $A$ and $B$?
- $A$ is a predication, $B$ is a conjunction
- *Renderings in $L^{\lambda}_{ar}$ preserve logical form*

# Propositional attitudes (work in progress)

Nixon claimed that he is not a crook

$$\xrightarrow{\text{formalize}} \text{Claimed}(\text{Nixon}, \text{not}(\text{crook}(\text{he})))$$

$$\xrightarrow{\text{coindex}} A \equiv \text{Claimed}(\dot{n}, \text{not}(\text{crook}(\dot{n}))) \text{ where } \{\dot{n} := \text{Nixon}\}$$

- The truth and meaning of $A$ depend on

  $\boxed{\text{the meaning of not}(\text{crook}(\dot{n})) \text{ when } \dot{n} \text{ refers to Nixon}}$

- Basic idea (close to Frege's):

  Referential intensions are (basically) tuples of functions in our universe $\mathfrak{M}_0$, and so $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ can talk about them
- Plan: make this precise and then (roughly) replace the attitudinal constant Claimed by a denotational constant which takes referential intensions as arguments
- The task is complicated by the free variable $\dot{n}$ in the box

Other examples of attitudinal constants:

say that ..., declare that ..., know that ..., believe that ...

# Syntax

- We add attitudinal constants of type

$$\mathrm{C} : \tilde{\mathrm{e}} \times \tilde{\mathrm{t}} \to \tilde{\mathrm{t}}$$

  (more complex ones are treated similarly)
- Extend the definition of terms: by

  | |
  |---|
  | *If C is an attitudinal constant,* $A : \tilde{\mathrm{e}}$ *and* $B : \tilde{\mathrm{t}}$ *is proper, then* $\mathrm{C}(A, B) : \tilde{\mathrm{t}}$ |

  ($B$ must be proper so it has a referential intension)
- Syncategorematic use of attitudinal constants:
  Claims by itself is not a term
- Nesting is allowed:

Dean expected that Nixon would claim that he is not a crook

$$\xrightarrow{\text{render}} \mathrm{Expected}(\mathrm{Dean}, \mathrm{Claims}(\dot{n}, \mathrm{not}(\mathrm{crook}(\dot{n})) \text{ where } \{\dot{n} := \mathrm{Nixon}\}))$$

# The key idea: Penelope's fears

Penelope thought that Ulysses was lost

$$\xrightarrow{\text{render}} \text{Thought}(\text{Penelope}, \underbrace{\text{lost}(\text{Ulysses})})$$

$$\text{int}(\text{lost}(\text{Ulysses})) = \text{int}(\text{lost}(\dot{u}) \text{ where } \{\dot{u} := \text{Ulysses}\})$$
$$= ((\dot{u}, \emptyset, \langle \dot{u} \rangle, \emptyset, \emptyset, (\alpha_0, \alpha_1))) = (\mathfrak{s}, (\alpha_0, \alpha_1)))$$

$$\alpha_0(\mathrm{g}, d) = \text{den}(\text{lost}(\dot{u}))(\mathrm{g}\{\dot{u} := d\}) = \text{lost}(d),$$
$$\alpha_1(\mathrm{g}) = \text{Ulysses}$$

- We can replace $\alpha_0, \alpha_1$ by

$$\alpha_0'(d) = \text{lost}(d), \quad \alpha_1' = \text{Ulysses}$$

set $\boxed{\text{int}'(\text{lost}(\text{Ulysses})) = (\mathfrak{s}, \text{lost}, \text{Ulysses})}$ and "unabbreviate"

- Thought(Penelope, lost(Ulysses))

$$:\equiv \text{Thought}^{\mathfrak{S}}(\text{Penelope}, \text{lost}, \text{Ulysses})$$

- with Thought$^{\mathfrak{S}}$ a suitably defined denotational constant

# Dealing with free variables: declaration of love (1)

Peter declares that he loves John's sister

$$\xrightarrow{\text{formalize}} \text{Declares}(\text{Peter}, \text{loves}(\text{he}, \text{sister}(\text{John})))$$

$$\xrightarrow{\text{coindex}} A \equiv \text{Declares}(\dot{p}, \text{loves}(\dot{p}, \text{sister}(\text{John}))) \text{ where } \{\dot{p} := \text{Peter}\}$$

$$\Rightarrow \text{Declares}\Big(\dot{p}, \text{loves}(\dot{p}, \dot{s}) \text{ where } \{\dot{s} := \text{sister}(j), j := \text{John}\}\Big) \quad \text{where } \{\dot{p} := \text{Peter}\}$$

$$\equiv \text{Declares}(\dot{p}, L) \text{ where } \{\dot{p} := \text{Peter}\}$$

with the abbreviation

$$L \equiv \text{loves}(\dot{p}, \dot{s}) \text{ where } \{\dot{s} := \text{sister}(j), j := \text{John}\} : \tilde{\mathrm{t}}.$$

- $\dot{p}$ is free in $L$ (quantifying in)
- Peter declares $L$ for a specific value of $\dot{p}$ (himself in this case)

Some man claims he loves Mary

$$\xrightarrow{\text{render}} \text{some}(\text{man})(\lambda(u)\text{Claims}(u, \text{loves}(u, \text{Mary})))$$

# Declaration of love (2)

$$L \equiv \text{loves}(\dot{p}, \dot{s}) \text{ where } \{\dot{s} := \text{sister}(j), j := \text{John}\} : \tilde{\mathrm{t}}$$

$$\mathfrak{s} = \text{shape}(L) = (\dot{s}, j, \langle \dot{p} \rangle, \langle \dot{s} \rangle, \emptyset, \langle j \rangle, \emptyset, \emptyset)$$

$$\text{system}(L) = (\alpha_0, \alpha_1, \alpha_2), \quad \text{int}(L) = (\mathfrak{s}, \text{system}(L)), \quad \text{with}$$

$$\alpha_0(\mathrm{g}, d_1, d_2) = \text{loves}(d_1, d_2), \quad \alpha_1(\mathrm{g}, d) = \text{sister}(d), \quad \alpha_2(\mathrm{g}) = \text{John}$$

We replace these by

$$\alpha_0'(d_1, d_2) = \text{loves}(d_1, d_2), \quad \alpha_1'(d) = \text{sister}(d), \quad \alpha_2' = \text{John}$$

which determine system($L$) **The crucial move:**

$$\text{Declares}(\dot{p}, L) :\equiv \text{Declares}^{\mathfrak{S}}(\dot{p}, \boxed{\text{fint}(L)})$$

$$\equiv \text{Declares}^{\mathfrak{S}}(\dot{p}, \boxed{\dot{p}, \lambda(p, s)\text{loves}(p, s), \lambda(\jmath)\text{sister}(\jmath), \text{John}})$$

fint($L$) is the formal referential intension of $L$ (a tuple of terms) and $\text{Declares}^{\mathfrak{S}}$ is a a denotational constant—defined empirically

# The $\lambda^{\mathrm{r}}$ operator

- To define formal referential intensions, we need to apply the $\lambda$-operator with recursive variables

$$\lambda^{\mathrm{r}}(x)(A) :\equiv \begin{cases} \lambda(x)(A), & \text{if } x \text{ is a pure variable,} \\ \lambda(x')(A\{x :\equiv x'\}), & \text{if } x \text{ is a recursion variable,} \end{cases}$$

where $x'$ is a fresh, pure variable of the same type as $x$, so

$$\lambda^{\mathrm{r}}(\dot{p}, \dot{s})\mathrm{loves}(\dot{p}, \dot{s}) \equiv \lambda(p, s)\mathrm{loves}(p, s)$$

- If $A$ is immediate or irreducible, then $\lambda^{\mathrm{r}}(x)(A)$ is also immediate or irreducible accordingly

# Formal referential intensions and the elimination of attitudinal constants (1)

$$\boxed{\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \ldots, \dot{p}_n := A_n\}}$$

- $\mathrm{FV}(A) = (x_1, \ldots, x_k) =$ the free variables of $A$
- $\vec{\mathbf{f}}_i(A) = (x_{s_1}, \ldots, x_{s_i}) = \mathrm{FV}(A) \cap \mathrm{FV}(A_i)$
- $\vec{\mathbf{r}}_i(A) = (\dot{p}_{t_1}, \ldots, \dot{p}_{t_i}) = \mathrm{FV}(A_i) \cap (\dot{p}_1, \ldots, \dot{p}_n)$
- $\mathfrak{s} = \mathrm{shape}(A) = (\dot{p}_1, \ldots, \dot{p}_n, \vec{\mathbf{f}}_0(A), \vec{\mathbf{r}}_0(A), \ldots, \vec{\mathbf{f}}_n(A), \vec{\mathbf{r}}_n(A))$
- $\alpha_i(u_{s_1}, \ldots, u_{s_i}, v_{t_1}, \ldots, v_{t_i})$
  $= \mathrm{den}(A_i)(\{x_{s_1} := u_{s_1}, \ldots, x_{s_i} := u_{s_i}, \dot{p}_{t_1} := v_{t_1}, \ldots, \dot{p}_{t_i} := v_{t_i}\}$
- $\boxed{\mathrm{int}'(A) = (\alpha_0, \ldots, \alpha_n)}$ (needs $\mathrm{shape}(A)$ to compute $\mathrm{den}(A)(\mathrm{g})$)
- $\boxed{\mathrm{fint}(A) \equiv (x_1, \ldots, x_k, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)A_0, \ldots, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)A_n)}$
  a sequence of formal terms
- $\mathrm{shape}(A)$ and $\mathrm{den}(\mathrm{fint}(A))$ determine $\mathrm{int}(A)$

$$\boxed{\mathrm{C}(B, A) :\equiv \mathrm{C}^{\mathrm{shape}(A)}(B, \mathrm{fint}(A))}$$

# Formal referential intensions and the elimination of attitudinal constants (2)

$$\text{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$$

$$\text{FV}(A) = (x_1, \dots, x_k) = \text{the free variables of } A$$

$$\boxed{\mathsf{C}(B, A) :\equiv \mathsf{C}^{\text{shape}(A)}(B, \text{fint}(A))}$$

$$\equiv \mathsf{C}^{\text{shape}(A)}(B, x_1, \dots, x_k, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)A_0, \dots, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)A_n)$$

$\overline{\text{Claims}}^{\mathfrak{s}}(y, u_1, \dots, u_k, r_0, r_1, \dots r_n)(a) = 1$

$\Longleftrightarrow$ there is an irreducible term

$$D \equiv D_0 \text{ where } \{\dot{p}_1 := D_1, \dots, \dot{p}_n := D_n\} : \tilde{\mathrm{t}}$$

with $\text{shape}(D) = \mathfrak{s}$ and such that

$$r_0 = \text{den}(\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)D_0) \dots, r_n = \text{den}(\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)D_n)$$

and in state $a$, $y$ claims $D$ for the values $x_1 := u_1, \dots, x_k := u_k$

# The form of formal referential intensions

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$$

$$\mathrm{FV}(A) = (x_1, \dots, x_k) = \text{the free variables of } A$$

$$\mathrm{fint}(A) \equiv (x_1, \dots, x_k, \underbrace{\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)A_0, \dots, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)A_n}_{\text{closed irreducible terms}})$$

- $A$ is strictly proper if every part $A_i$ is proper
- Not strictly proper: $\dot{p}$ where $\{\dot{p} := \dot{q}, \dot{q} := \mathrm{John}\}$ (umm ... John)
- *Natural language phrases are rendered by strictly proper terms* (?)
- If $A$ is strictly proper and $C$ is attitudinal, then

$$\begin{aligned} \mathrm{C}(B, A) &:\equiv \mathrm{C}^{\mathrm{shape}(A)}(B, \mathrm{fint}(A)) \\ &\equiv \mathrm{C}^{\mathrm{shape}(A)}(B, x_1, \dots, x_k, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)A_0, \dots, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)A_n) \\ &\Rightarrow \mathrm{C}^{\mathrm{shape}(A)}(B, x_1, \dots, x_k, \overrightarrow{\dot{p}}) \text{ where } \{\overrightarrow{\dot{p}} := \overrightarrow{\lambda^{\mathrm{r}}(\vec{\mathbf{f}}, \vec{\mathbf{r}})A}\} \end{aligned}$$

# Attitudes on propositions with free variables

$$A \equiv \mathsf{Claims}(\mathsf{Dean}, \mathsf{crook}(\mathsf{Nixon}))$$
$$\equiv \mathsf{Claims}^{\mathfrak{s}_1}(\mathsf{Dean}, \mathsf{crook}, \mathsf{Nixon})$$

$$\mathfrak{s}_1 = \mathsf{shape}(\mathsf{crook}(\mathsf{Nixon})) = \mathsf{shape}(\mathsf{crook}(\dot{n}) \text{ where } \{\dot{n} := \mathsf{Nixon}\})$$

$$B \equiv \mathsf{Claims}(\mathsf{Dean}, \mathsf{crook}(\dot{n})) \text{ where } \{\dot{n} := \mathsf{Nixon}\}$$
$$\equiv \mathsf{Claims}^{\mathfrak{s}_2}(\mathsf{Dean}, \dot{n}, \mathsf{crook}) \text{ where } \{\dot{n} := \mathsf{Nixon}\}$$

$$\mathfrak{s}_2 = \mathsf{shape}(\mathsf{crook}(\dot{n}))$$

- No assumptions are made on whether $\mathfrak{M}_0 \models A = B$ or $A \approx B$
- Do we have intuitions about "claiming open propositions"?
- Could the answers be different for "Claims" and for "Believes", so that they are not a matter of logic?

$$\boxed{\begin{aligned} (?)\ \ & \mathsf{Claims}(x, \mathsf{crook}(\dot{n}))(a) \text{ when } \dot{n} \text{ refers to } \mathsf{N} \\ & \qquad \sim \mathsf{Claims}(x, \mathsf{crook}(\mathsf{He}))(a) \text{ with } \mathsf{He}(a) = \mathsf{N}(a) \end{aligned}}$$

# Coindexing of meanings

Peter claims that Mary likes John but Mary denies it

$$\xrightarrow{\text{formalize}} \text{Claims}(\text{P}, \underline{\text{likes}(\text{M}, \text{J})}) \text{ but Denies}(\text{M}, \underline{\text{it}})$$

$$\equiv \text{Claims}^{\mathfrak{S}_1}(\text{P}, \underline{\text{likes}, \text{M}, \text{J}}) \text{ but Denies}(\text{M}, \underline{\text{it}})$$

$$\xrightarrow{\text{coindex}} \text{Claims}^{\mathfrak{S}_1}(\text{P}, \dot{l}, \dot{m}, \dot{j}) \text{ but Denies}^{\mathfrak{S}_1}(\text{M}, \dot{l}, \dot{m}, \dot{j})$$

$$\text{where } \{\dot{l} := \text{likes}, \dot{m} := \text{M}, \dot{j} := \text{J}\}$$

$$\mathfrak{s}_1 = \text{shape}(\text{likes}(\text{M}, \text{J})) \qquad \mathfrak{s}_2 = \text{shape}(\text{likes}(\dot{m}, \text{J}))$$

Peter claims that Mary likes John but she denies it

$$\xrightarrow{\text{formalize}} \text{Claims}(\text{P}, \underline{\text{likes}(\text{M}, \text{J})}) \text{ but Denies}(\text{she}, \underline{\text{it}})$$

$$\xrightarrow{\text{coindex}} \text{Claims}(\text{P}, \underline{\text{likes}(\dot{m}, \text{J})}) \text{ but Denies}(\dot{m}, \underline{\text{it}}) \text{ where } \{\dot{m} := \text{M}\}$$

$$\equiv \text{Claims}^{\mathfrak{S}_2}(\text{P}, \underline{\dot{m}, \text{likes}, \text{J}}) \text{ but Denies}(\dot{m}, \underline{\text{it}})$$

$$\xrightarrow{\text{coindex}} \text{Claims}^{\mathfrak{S}_2}(\text{P}, \dot{m}, \dot{l}, \dot{j}) \text{ but Denies}^{\mathfrak{S}_2}(\dot{m}, \dot{m}, \dot{l}, \dot{j})$$

$$\text{where } \{\dot{l} := \text{likes}, \dot{m} := \text{M}, \dot{j} := \text{J}\}$$

# Afterward

- Better title: $\boxed{\text{A theory of logical meaning and synonymy}}$

- The language $L^{\lambda}_{ar}$: an extension of Montague's language of intensional logic with assignments, using ideas from programming languages
- Mathematical modeling of meanings and synonymy, and the development of a (suitably) complete logic for synonymy
- Logical form $\sim$ canonical form
- New tools to coindex, with novel results
- A (quasi)-Fregean modeling of propositional attitudes (preliminary)

Missing:

- Utterances and local meanings (in the manuscript)
- Factual content (Eleni Kalyvianaki)
- Vagueness: "approximate synonymy"