English as a programming language?

Yiannis N. Moschovakis
UCLA and University of Athens

ESSLLI, 21 July, 2009

cf. Richard Montague (1970) *English as a formal language*

Everything is not like everything else!

1960 Seminar on the *Theory of Knowledge* with the authors of

A Logical Calculus of the Ideas Immanent in Nervous Activity,
by Warren McCulloch and Walter Pitts, 1943

Student: Isn't this very much like that, which we studied last week?

McCulloch: Many things are like many other things—and it is useful to note this when we first encounter them;
to understand a phenomenon more deeply, however, you should focus on how it differs from similar phenomena

Lesson: Not everything is like everything else!

$\mathcal{Q}$: Is there a substantial common (or similar) important feature of natural and formal or programming languages
which helps us understand them better?

How I got into this business

Scott's denotational semantics for programming languages:

$$\begin{tikzcd} \text{program } P \arrow[r, shift left, "\text{den}"] \arrow[r, shift right, Rightarrow, "\text{impls}"'] & \text{value}(P) \end{tikzcd}$$

Correct implementations compute the independent denotation

$$\boxed{\text{Tarski-type value conditions } \underline{\text{vs}} \text{ implementation rules}}$$

<u>A program expresses an algorithm</u>; where is it?

$$\begin{tikzcd} \text{program } P \arrow[r] & \text{algorithm}(P) \arrow[r, shift left, "\text{den}"] \arrow[r, shift right, Rightarrow, "\text{impls}"'] & \text{value}(P) \end{tikzcd}$$

$\mathfrak{Frege}$: sentence $A \longrightarrow$ sense$(A) \longrightarrow$ den(A)

The talk in slogan form

$\boxed{\textit{The meaning of a term is the algorithm which computes its denotation}}$

Meanings are algorithms?

It depends on what the meaning of the word 'is' is

In mathematics, we say:

$\underline{\text{A point in the plane is a pair } (x, y) \text{ of real numbers}}$

(and worse, $(x, y) = \{\{x\}, \{x, y\}\}$!)

but Euclid did OK without "knowing" this!

$\underline{\text{The plane can be faithfully modeled by the Cartesian product } \mathbb{R} \times \mathbb{R}}$

and

$\boxed{\text{Meanings can be faithfully modeled by algorithms}}$

(meaning abstract, not necessarily implementable algorithms)

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

Outline

(1) Formal Fregean semantics
(2) Meaning and synonymy
(3) What are the objects of belief? (Local synonymy)
(4) The decision problem for synonymy

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/∼ynm

The methodology of formal Fregean semantics

- An *interpreted formalized language* L is selected
- The rendering operation on a fragment of natural language:

$$\text{natural language expression} + \text{informal context} \xrightarrow{\text{render}} \text{formal expression} + \text{state}$$

- Semantic values (denotations, meanings, etc.) are defined rigorously for the formal expressions of L and assigned to natural language expressions via the rendering operation
- Montague: L *should be a higher type language*
  (to model co-ordination, co-indexing, . . .)
- Claim: L *should have programming constructs*
  (to model co-indexing, self-reference, meanings, . . .)

The typed λ-calculus with recursion $\mathsf{L}^{\lambda}_{r}(K)$ - types

An extension of the typed λ-calculus, into which Montague's Language of Intensional Logic LIL can be easily interpreted (Gallin)

Basic types $b \equiv e \mid t \mid s$ (entities, truth values, states)

Types: $\sigma :\equiv b \mid (\sigma_1 \to \sigma_2)$

Abbreviation: $\sigma_1 \times \sigma_2 \to \tau \equiv (\sigma_1 \to (\sigma_2 \to \tau))$

Every non-basic type is uniquely of the form

$$\sigma \equiv \sigma_1 \times \cdots \times \sigma_n \to b$$

The typed λ-calculus with recursion $\mathsf{L}^\lambda_r(K)$ - syntax

Constants: A finite set K of typed constants (run, cow, he, the, every)
he : $(s \to e)$

⋆ *Pure variables*: $v_0^\sigma, v_1^\sigma, \ldots,$ for each type σ $(v : \sigma)$
⋆ *Recursive variables*: $p_0^\sigma, p_1^\sigma, \ldots,$ for each type σ $(p : \sigma)$

Terms – with assumed type restrictions and assigned types $(A : \sigma)$

$$A :\equiv v \mid p \mid c \mid B(C) \mid \lambda(v)(B)$$
$$\star \mid A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\}$$

$$C : \sigma, B : (\sigma \to \tau) \implies B(C) : \tau$$
$$v : \sigma, B : \tau \implies \lambda(v)(B) : (\sigma \to \tau)$$
$$A_0 : \sigma \implies A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} : \sigma$$

Abbreviation: $A(B, C, D) \equiv A(B)(C)(D)$

$\mathsf{L}^{\lambda}_{r}(K)$ - denotational semantics

- We are given basic sets $\mathbb{T}_s, \mathbb{T}_e$ and $\mathbb{T}_t \subseteq \mathbb{T}_e$ for the basic types

$$\begin{aligned} \mathbb{T}_{\sigma\to\tau} &= \text{the set of all functions } f : \mathbb{T}_\sigma \to \mathbb{T}_\tau \\ \mathbb{P}_b &= \mathbb{T}_b \cup \{\bot\} = \text{the "flat poset" of } \mathbb{T}_b \\ \mathbb{P}_{\sigma\to\tau} &= \text{the set of all functions } f : \mathbb{T}_\sigma \to \mathbb{P}_\tau \end{aligned}$$

  - v_i^σ varies over $\mathbb{T}_\sigma$, the total objects;
  - p_i^σ varies over $\mathbb{P}_\sigma$, the partial objects

$\mathbb{T}_\sigma \subseteq \mathbb{P}_\sigma$ and $\mathbb{P}_\sigma$ is a complete poset (with the pointwise ordering)

- We are given an object $c : \mathbb{P}_\sigma$ for each constant $\mathsf{c} : \sigma$
  - If $A : \sigma$ and π is a type-respecting assignment to the variables, then $\mathrm{den}(A)(\pi) \in \mathbb{P}_\sigma$ (values are partial objects)
  - Recursive terms are interpreted by the taking of least-fixed-points

Rendering natural language in $\mathsf{L}^{\lambda}_{r}(K)$

$$\tilde{t} \equiv (s \to t) \qquad \text{(type of Carnap intensions)}$$

$$\tilde{e} \equiv (s \to e) \qquad \text{(type of individual concepts)}$$

$$\text{Abelard loves Eloise} \xrightarrow{\text{render}} \text{loves(Abelard,Eloise)} : \tilde{t}$$

$$\text{Obama is the president} \xrightarrow{\text{render}} \text{eq(Obama,the(president))} : \tilde{t}$$

$$\text{He is the president} \xrightarrow{\text{render}} \text{eq(He,the(president))} : \tilde{t}$$

$$\text{liar} \xrightarrow{\text{render}} p \text{ where } \{p := \neg p\} : t$$

$$\text{truthteller} \xrightarrow{\text{render}} p \text{ where } \{p := p\} : t$$

$$\text{Abelard}, \text{Eloise}, \text{Obama}, \text{He} : \tilde{e}$$

$$\text{president} : \tilde{e} \to \tilde{t}, \text{eq} : \tilde{e} \times \tilde{e} \to \tilde{t}$$

$$\neg : t \to t, \text{the} : (\tilde{e} \to \tilde{t}) \to \tilde{e}$$

$$\text{den(liar)} = \text{den(truthteller)} = \bot$$

Co-ordination and co-indexing in $\mathsf{L}^{\lambda}_{r}(K)$

John stumbled and John fell $\xrightarrow{\text{render}}$ stumbled(John) & fell(John)
(conjunction)

John stumbled and fell $\xrightarrow{\text{render}}$ $\lambda(x)\Big(\text{stumbled}(x)\ \&\ \text{fell}(x)\Big)(\text{John})$
(predication after co-ordination)

These are in Montague's LIL (as it is interpreted in $\mathsf{L}^{\lambda}_{r}(K)$)

John stumbled and he fell $\xrightarrow{\text{render}}$ stumbled(j) & fell(j) where $\{j := \text{John}\}$
(conjunction after co-indexing)

The logical form of this sentence <u>cannot</u> be captured faithfully in LIL — recursion models co-indexing preserving logical form

Can we say nonsense in $\mathsf{L}_r^\lambda(K)$?

Yes!
In particular, we have variables over states—so we can explicitly refer to the state (even to two states in one term); LIL does not allow this, *because we cannot do this in English*

Consider the terms

$$A \equiv \mathrm{rapidly}(\mathrm{tall})(\mathrm{John}),\ \ B \equiv \mathrm{rapidly}(\mathrm{sleeping})(\mathrm{John}) : \tilde{t}$$

A and B are terms of LIL,

not the renderings of correct English sentences

- The target formal language is a tool for defining rigorously the desired semantic values and it needs to be richer than a direct formalization of the relevant fragment of natural language

  —to insure compositionality, if for no other reason

Meaning and synonymy in $\mathsf{L}^{\lambda}_{r}(K)$

- For a sentence $A : \tilde{t}$, the Montague sense of A is $\mathrm{den}(A) : \mathbb{T}_s \to \mathbb{T}_t$, so that

  there are infinitely many primes

  is Montague-synonymous with $1 + 1 = 2$

- In $\mathsf{L}^{\lambda}_{r}(K)$: *The meaning of a term A is modeled by an algorithm* $\mathrm{refint}(A)$ *which computes den*$(A)(\pi)$ *for every* π
- The referential intension $\mathrm{refint}(A)$ is compositionally determined from A
- $\mathrm{refint}(A)$ is an abstract (not necessarily implementable) recursive algorithm of $\mathsf{L}^{\lambda}_{r}(K)$
- Referential synonymy: $\boxed{A \approx B \iff \mathrm{refint}(A) \sim \mathrm{refint}(A)}$

Reduction, Canonical Forms and the Synonymy Theorem

- A reduction relation $A \Rightarrow B$ is defined on terms of $\mathsf{L}_r^\lambda(K)$ (B expresses more simply the meaning of A 5 trivial + 5 important clauses or meaning postulates)
- *CF Theorem*. Each term A is reducible to a unique (up to congruence) irreducible recursive term, its canonical form

$$A \Rightarrow \mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \dots, p_n := A_n\}$$

- $\boxed{\mathrm{refint}(A) = (\mathrm{den}(A_0), \mathrm{den}(A_1), \dots, \mathrm{den}(A_n))}$
- The parts $A_0, \dots, A_n$ of A are irreducible, explicit terms
- $\mathrm{cf}(A)$ models the logical form of A
- *Synonymy Theorem. $A \approx B$ if and only if*

$$B \Rightarrow \mathrm{cf}(B) \equiv B_0 \text{ where } \{p_1 := B_1, \dots, p_m := B_m\}$$

so that $n = m$ and for $i \leq n$, $den(A_i) = den(B_i)$

Is this notion of meaning Fregean?

Evans (in a discussion of Dummett's similar, computational interpretations of Frege's sense):

> *"This leads [Dummett] to think generally that the sense of an expression is (not a way of thinking about its [denotation], but) a method or procedure for determining its denotation. So someone who grasps the sense of a sentence will be possessed of some method for determining the sentence's truth value*
> *. . . ideal verificationism*
> *. . . there is scant evidence for attributing it to Frege"*

Converse question: For a sentence A, if you possess the method determined by A for determining its truth value, do you then "grasp" the sense of A?

(Sounds more like Davidson rather than Frege)

Some referential synonymies and non-synonymies

- There are infinitely many primes $\not\approx 1 + 1 = 2$
- $A \,\&\, B \approx B \,\&\, A$
- The morning star is the evening star

  $\approx$The evening star is the morning star

  (This fails with Montague's renderings)
- Abelard loves Eloise $\approx$ Eloise $\boxed{\text{is loved by}}$ Abelard (Frege)
- $2 + 3 = 6 \approx 3 + 2 = 6$ (with $+$ and the numbers primitive)
- liar $\not\approx$ truthteller
- John stumbled and he fell $\xrightarrow{\text{render}}$

  $A \equiv \text{stumbled}(j) \,\&\, \text{fell}(j) \text{ where } \{j := \text{John}\}$

  A is not $\approx$ with any *explicit* term (including any term from LIL)

Utterances, local meanings, local synonymy

An utterance is a pair (A, u), where A is a sentence, $A : \tilde{t}$ and u is a state.

We add to the language (for convenience only) a parameter $\bar{u}$ for each state u and express (A, u) by the term $A(\bar{u})$

The local meaning of A at the state u is $\mathrm{refint}(A(\bar{u}))$

$$A \approx_u B \iff A(\bar{u}) \approx B(\bar{u})$$

Obama is the president$(\bar{u})$

$\Rightarrow_{cf} \mathrm{eq}(b)(L)(\bar{u})$ where $\{b := \mathrm{Obama}, L := \mathrm{the}(p), p := \mathrm{president}\}$

He is the president$(\bar{u})$

$\Rightarrow_{cf} \mathrm{eq}(b)(L)(\bar{u})$ where $\{b := \mathrm{He}, L := \mathrm{the}(p), p := \mathrm{president}\}$

Obama is the president $\not\approx_u$ He is the president

even if at the state $\bar{u}$, $\mathrm{He}(\bar{u}) := \mathrm{Obama}(\bar{u})$

Three aspects of meaning for a sentence $A : \tilde{t}$

Referential intension	$\text{refint}(A)$	Referential synonymy $\approx$
Local meaning at u	$\text{refint}(A(\bar{u}))$	Local synonymy $\approx_u$
Factual content at u	$\text{FC}(A, u)$	Factual synonymy $\approx_{f,u}$

The *factual content* of a sentence at a state u gives a *representation of the world* at u (Eleni Kalyvianaki's Ph.D. Thesis)

Obama is the president $\not\approx_u$ He is the president

Obama is the president $\approx_{f,u}$ He is the president

Claim: *The objects of belief are local meanings*

The distinction between local meaning and factual content are related to David Kaplan's distinction between the *character* and *content* of a sentence at a state

Is referential synonymy decidable?

Synonymy Theorem. $A \approx B$ if and only if

$$A \Rightarrow \mathrm{cf}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \dots, p_n := A_n\}$$
$$B \Rightarrow \mathrm{cf}(B) \equiv B_0 \text{ where } \{p_1 := B_1, \dots, p_n := B_n\}$$

so that for $i = 0, \dots, n$ and all π, $den(A_i)(\pi) = den(B_i)(\pi)$.

- Synonymy is reduced to denotational equality for explicit, irreducible terms (the "truth conditions" of A)
- Denotational equality for arbitrary terms is undecidable (there are constants, with fixed interpretations)
- The explicit, irreducible terms are very special

  — but by no means trivial!

The synonymy problem for $\mathsf{L}^{\lambda}_{r}(K)$ (with finite K)

- The decision problem for $\mathsf{L}^{\lambda}_{r}(K)$-synonymy is open

Theorem *If the set of constants K is finite, then synonymy is decidable for terms of* **adjusted level** ≤ 2

These include terms constructed "simply" from

Names of "pure" objects	$0, 1, 2, \emptyset, \dots$	$: e$
Names, demonstratives	John, I, he, him	$: \tilde{e}$
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, loves, be	$: \tilde{e} \times \tilde{e} \to \tilde{t}$
Adverbs	rapidly	$: (\tilde{e} \to \tilde{t}) \to (\tilde{e} \to \tilde{t})$

Proof is by reducing this claim to the Main Theorem in the 1994 paper (for a corrected version see www.math.ucla.edu/∼ynm)

Explicit, irreducible identities that must be known

- Los Angeles = LA   (Athens = Αθήνα)
- $x \mathbin{\&} y = y \mathbin{\&} x$
- $\text{between}(x, y, z) = \text{between}(x, z, y)$
- $\text{love}(x, y) = \text{be\_loved}(y, x)$

A dictionary is needed—but what kind and how large?

$$\text{ev}_2(\lambda(u_1, u_2)r(u_1, u_2, \vec{a}), b, z) = \text{ev}_1(\lambda(v)r(v, z, \vec{a}), b)$$

Evaluation functions: both sides are equal to $r(b, z, \vec{a})$

The dictionary line which determines this is (essentially)

$$\lambda(s)x(s, z) = \lambda(s)y(s) \implies \text{ev}_2(x, b, z) = \text{ev}_1(y, b)$$

The form of the decision algorithm

- A finite list of true dictionary lines is constructed, which codifies the relationships between the constants
- Given two explicit, irreducible terms A, B of adjusted level ≤ 2, we construct (effectively) a finite set $L(A, B)$ of lines such that

$$\begin{aligned} &\models A = B \\ \iff &\text{ every line in } L(A, B) \text{ is congruent to one in the dictionary} \end{aligned}$$

- It is a lookup algorithm, justified by a finite basis theorem
- Complexity: NP; the graph isomorphism problem is reducible to the synonymy problem for very simple (propositional) recursive terms