Recursion in higher types and the semantics of natural languages

by

Yiannis N. Moschovakis

Department of Mathematics
University of California, Los Angeles
and University of Athens
www.math.ucla.edu/~ynm

D. A. Martin Conference
Berkeley, May 27–28, 2001

The classical type structure

Types : $\sigma = e \mid (\sigma_1 \to \sigma_2)$

Spaces:

$$\begin{aligned}\mathbb{T}_e &= \text{some fixed, infinite set,} \\ \mathbb{T}_{(\sigma\to\tau)} &= (\mathbb{T}_\sigma \to \mathbb{T}_\tau) = {}^{\mathbb{T}_\sigma}\mathbb{T}_\tau\end{aligned}$$

Assume $0, 1 \in \mathbb{T}_e$, $\mathbb{N} \subseteq \mathbb{T}_e$, etc.

• In the applications to natural languages, there is also a set I of **possible worlds** (contexts) in the background.

The typed λ-calculus $\boldsymbol{\lambda}(\vec{c}) = \boldsymbol{\lambda}(c_1, \dots, c_K)$

Typed variables : $v_0^\sigma, v_1^\sigma, \dots$

$x : \sigma$ stands for "x is a variable of type σ"

Typed constants : $c_i : \sigma_1^i \times \cdots \times \sigma_{n_i}^i \to e$

Typed terms:

$$\begin{aligned} A \ :\equiv \ & x(y) \mid x(A) \mid A(x) \mid A(B) \mid \lambda x(A) \\ & \mid c(A_1, \dots, A_n) \mid \exists_\sigma(\lambda x^\sigma A) \mid \forall_\sigma(\lambda x^\sigma A) \end{aligned}$$

- Variables are not terms!
  (but for higher types, $x = \lambda u x(u)$)

- Abbreviations:

$$\begin{aligned} A \ \&\ B \ &:\equiv \ \&(A, B) \qquad (\& \text{ is some } c_i), \\ \exists x A \ &:\equiv \ \exists_\sigma(\lambda x^\sigma A) \end{aligned}$$

Semantics for $\boldsymbol{\lambda}(\vec{c})$

Models: $\mathcal{T}(\vec{c}) = (\mathbb{T}_e, c_1, \ldots, c_K)$, with

$$c_i : \mathbb{T}_{\sigma_1^i} \times \cdots \times \mathbb{T}_{\sigma_{n_i}^i} \to \mathbb{T}_e$$

• For each (type-respecting) assignment π to the variables, we have a value, such that

$$\mathsf{type}(A) = \sigma \implies \mathsf{den}(A, \pi) \in \mathbb{T}_\sigma.$$

• If we have a set I of contexts, then each c_l may also depend on the context, and the basic semantic relation takes the form

$$\mathsf{type}(A) = \sigma, i \in I \implies \mathsf{den}(A, \pi, i) \in \mathbb{T}_\sigma.$$

Adding recursion—the language $\boldsymbol{\lambda}_r(\vec{c})$

Terms:

$$\begin{aligned} A \quad :\equiv \quad & x(y) \mid x(A) \mid A(x) \mid A(B) \mid \lambda x(A) \\ & \mid c(A_1, \ldots, A_n) \mid \exists x(A) \mid \forall x(A) \\ & \mid A_0 \text{ where } \{x_1 = A_1, \ldots, x_n = A_n\} \end{aligned}$$

- $x_1, \ldots, x_n$ are bound

$$\begin{aligned} &\mathrm{den}(A_0 \text{ where } \{x_1 = A_1, \ldots, x_n = A_n\}, \pi) \\ &\qquad = \mathrm{den}(A_0, \pi\{x_1 := \bar{y}_1, \ldots, x_n = \bar{y}_n\}) \end{aligned}$$

where:

$\bar{y}_1, \ldots, \bar{y}_n$ are the (canonical) solutions of the recursive system

$$y_i = \mathrm{den}(A_i, \pi\{x_1 := y_1, \ldots, x_n = y_n\}) \quad (i = 1, \ldots, n)$$

- Recursive systems cannot be solved in the classical type structure.

Partial objects—the Kleene type structure

- $(A \rightharpoonup B) =$ all partial functions on A to B

- $\mathbb{P}_e = \mathbb{T}_e \cup \{\perp\}$ (with $\perp$ *objectifying the undefined* in Scott's terminology)

- $\mathbb{P}_{(\sigma\to\tau)} = (\mathbb{T}_\sigma \rightharpoonup \mathbb{P}_\tau) \supseteq \mathbb{T}_{(\sigma\to\tau)}$.

- Each $\mathbb{P}_\sigma$ is a **chain-complete partial ordered set**, and so systems of monotone recursive equations have least-fixed-points.

- In the Platek model, we take the hereditarily monotone objects,

$$\mathbb{P}_{(\sigma\to\tau)} = \text{all monotone functions } p : \mathbb{P}_\sigma \to \mathbb{P}_\tau$$

Semantics for $\boldsymbol{\lambda}_r(\vec{c})$ in $\mathcal{T}(\vec{c})$

- Variables range over partial objects

- Each c_i in the structure $\mathcal{T}(\vec{c})$ is a **monotone functional** on partial objects to partial objects.

- $$\exists_\sigma(p) = \begin{cases} 1, & \text{if } (\exists x \in \mathbb{T}_\sigma)[p(x) = 1], \\ 0, & \text{if } (\forall x \in \mathbb{T}_\sigma)[p(x) = 0], \\ \uparrow & \text{otherwise.} \end{cases}$$

- Application is interpreted strictly
  i.e., $p(x) \uparrow$ if x is not a total object.

- Recursion is by least-fixed-points.

- The Kleene-recursive functionals in $\mathcal{T}(\vec{c})$ are those defined by terms of $\boldsymbol{\lambda}_r(\vec{c})$.

Decidability questions

Theorem (H. Friedman, 1973) The decision problem for (semantic) identities

$$A = B$$

between pure terms (no constants) is decidable.

- Trivially, one cannot decide identities of the form

$$f(S^{(n)}(0)) = 1$$

for some fixed $f : \mathbb{N} \to \mathbb{N}$ and $S(x) = x + 1$

Terms with constants

Proposition (with JRM). For every $S \subseteq \mathbb{N}$, there is a functional $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ such that for all prime m,

$$m \notin S \iff \lambda p\, f(p^{(m)}) = \lambda p\, 1.$$

(So identities of this form are not decidable.)

Proof. Take (for total q)

$$f(q) = \begin{cases} 0 & \text{if } q \text{ is 1-1 and for some prime} \\ & m \in S, \quad q[\mathbb{N}] = [m, \infty), \\ 1, & \text{otherwise} \end{cases}$$

Irreducible (depth=2) terms

- Abbreviations: $p(B, C, D) \equiv p(B)(C)(D)$, $\lambda u(\lambda v(A)) \equiv \lambda uv(A)$.

Immediate expressions ($p, x_1, \ldots, x_n$ variables):

$$x, \quad p(x_1, \ldots, x_n), \quad \lambda\vec{u}\ p(x_1, \ldots, x_n)$$

Irreducible terms: immediate and

$$p(A_1, \ldots, A_n), \quad \lambda\vec{u}\ p(A_1, \ldots, A_n)$$
$$f(A_1, \ldots, A_n), \quad \lambda\vec{u}\ f(A_1, \ldots, A_n)$$

where $A_1, \ldots, A_n$ are immediate and f is any constant—$c_i, \exists, \forall$.

Irreducible identities:

$$f(x, x, y) = g(y, x, y), \quad \lambda xy\ f(x, x, y) = \lambda xy\ g(y, x, y)$$
$$f(\lambda st\ x(s, t)) = g(\lambda st\ x(t, s))$$
$$f(\lambda x\ x(y, u)) = g(\lambda s\lambda t\ r(x, y, \ldots))$$
$$f(\lambda u\ p(x, y, x, \ldots)) = p(u, x, \lambda s\ q(s, c, \ldots), \ldots)$$

The main result

Theorem For each structure $\mathcal{T}(\vec{c})$, the decision problem for irreducible identities in $\mathcal{T}(\vec{c})$ is decidable.

Similarly, for any class of structures

$$\{\mathcal{T}(\vec{c}) \mid \vec{c} \in \mathcal{C}\},$$

the decision problem for the relation

$$\alpha \sim_{\mathcal{C}} \beta \iff (\forall \vec{c} \in \mathcal{C})[\mathcal{T}(\vec{c}) \models \alpha = \beta]$$

between irreducible terms is decidable.

- The proof constructs for each $\mathcal{C}$ a finite **dictionary** of all the **lines** ("sequents" of a certain shape) which are true in every $\mathcal{T}(\vec{c})$ with $\vec{c} \in \mathcal{C}$; it is shown that each identity is equivalent to a finite number of lines, effectively computable from it; and so one simply checks if these lines occur in the dictionary.

- The complexity is linear (basically a "lookup" algorithm)

Frege's Sense and Denotation (1892)

$$A \longrightarrow \text{sense}(A) \longrightarrow \text{den}(A)$$

- Terms 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 $A \mapsto \text{sense}(A)$ is **compositional**, i.e., the meaning of a term is determined by the meanings of its subterms

---

$$9 + 4 = 13$$

There are infinitely many prime numbers

(It is not specified what kinds of objects senses are.)

Pushing meanings aside

$$\begin{tikzcd} A \arrow[r] \arrow[rr, bend right] & \text{sense}(A) \arrow[r] & \text{den}(A) \end{tikzcd}$$

- Direct **compositionality principle for denotations**

- Makes possible the development of denotational semantics (model theory), which leave meanings to the "pre-theoretic" level

- Slogan: the semantics of a language comprise the rules which determine the function

$$A \mapsto \text{den}(A)$$

(and perhaps only for sentences A).

Problems with meaning-free semantics

Indirect reference

- 'He believes that she has taken a lover'

Frege postulates that *in this indirect context*,

$$\text{den}(\text{'she has taken a lover'}) = \text{sense}(\text{'she has taken a lover'})$$

but then we need

$\text{sense}_2(A) =$ the sense of A in indirect contexts

- 'She knows that he believes she has taken a lover'

- The proliferation of denotations and senses makes the scheme unworkable, and few accept it now.

Problems with meaning-free semantics

Translation

- Faithful translation should **preserve meaning**

$$
\begin{tikzcd}
A \arrow[r] \arrow[d, "\text{translation}"'] & \text{sense}(A) \arrow[r] & \text{den}(A) \\
B \arrow[ur] & &
\end{tikzcd}
$$

- How can we define the notion of **faithful translation** between two languages, using only the function?

$$A \mapsto \text{den}(A)$$

- Standard definitions of translation from L_1 to L_2 are via homomorphisms from (the syntax of) L_1 to L_2, and assume great (if not complete) similarity of grammar.

What are meanings?

Russell propositions

RP('John loves Mary') = **J** den('loves') **M**

RP('Mary is loved by John')
= **M** den('is loved by') **J**

RP('John and Mary love bicycling')
= **J** den('and') **M** den('love bicycling')

RP('Mary and John love bicycling')
= **M** den('and') **J** den('love bicycling')

(**J** and **M** are **the persons**)

- Example of **structured meanings**

- The first two and the last two examples above are not **synonymous** (Synonymy too fine)

- Church's approaches are even finer—meanings are essentially syntactical objects

What are meanings?

Carnap, Montague: Carnap intensions

• Let I = the set of **contexts** (possible worlds), so that for each $i \in I$ we have a (possibly different) denotation,

$$\text{den}(A)(i) = \text{the denotation of } A \text{ in } i$$

• $\text{CI}(A) = \text{den}(A) = \lambda i \ \text{den}(A)(i)$.

• Montague adopts this in the first serious attempt to give a "mathematical" treatment of the semantics of fragments of English (1970)

• All true mathematical sentences have the same meaning (Synonymy too coarse)

• Carnap intensions are unlikely objects of belief

Problems in a meaningful semantics

$$\begin{tikzcd} A \arrow[dr] & & \\ B \arrow[r] & \text{meaning} \arrow[r] & \text{denotation} \\ C \arrow[ur] & & \end{tikzcd}$$

- Different terms (perhaps in different languages) may have the same meaning

- What are meanings?

- What is the relation of synonymy (faithful translation) between meanings?

- What kind of operations on meanings are **propositional attitudes** — like **knowledge** or **belief**?

How the higher types come in

- Taking compositionality seriously—**parsing**

John loves every blond woman

$$\begin{aligned} &= \underline{\text{loves}}(\text{John, every blond woman}) \\ &= \underline{\text{loves}}(\text{John},\underline{\text{every}}(\text{blond},\text{woman})) \\ &= \underline{\text{loves}}(\text{John},\underline{\text{every}}(\underline{\text{blond}}(\text{woman}))) \end{aligned}$$

- The underlined words denote functions

- The types of <u>loves</u> and <u>every</u> are quite complex.

John loves himself $= \Big(\lambda \text{x } \underline{\text{loves}}(\text{x,x})\Big)(\text{John})$

- Montague (essentially) claimed that substantial fragments (if not all) of English can be "expressed faithfully" in $\boldsymbol{\lambda}(\vec{c})$ with a suitable (very large) $\vec{c}$.

Programming languages

- Operational semantics

$$\text{Program } A \xrightarrow{\text{impl}} \text{value}(A)$$

- To prove (or just express) that an implementation is correct you need an independent definition of the correct output of a program

- Denotational semantics
(Scott-Strachey, 1971)

$$\begin{tikzcd}[column sep=huge] \text{Program } A \arrow[r] \arrow[r, bend left=60, "\text{implementations}"] \arrow[r, bend left=35] & \text{value}(A) \end{tikzcd}$$

Where are the algorithms?

$$\begin{tikzcd} A \arrow[dr] & & \\ B \arrow[r] & \text{algorithm} \arrow[r] & \text{denotation} \\ C \arrow[ur] & & \end{tikzcd}$$

- A program expresses an algorithm

- Different programs (even in different programming languages) may express the same algorithm

- terms – programs

- meanings – algorithms

- (Implementations are deliberately left out of this picture)

What are algorithms?

- Algorithms are constructed using calls to given function(al)s; application; conditional definitions; and recursion

- One must understand recursive definitions independently of implementations; take it as primitive

- The **fixed-point theory of programs** (Scott, Park, ...)

Recursors

• A W-**recursor** (with possible output in a set W) is a tuple

$$\alpha = (\alpha_0, \alpha_1, \dots, \alpha_n)$$

of monotone mappings where, for suitable chain-complete posets $D_1, \dots, D_n$,

$$\begin{gathered} \alpha_0 : D_1 \times \cdots \times D_n \rightharpoonup W, \\ \alpha_i : D_1 \times \cdots \times D_n \to D_i, \quad (i = 1, \dots, n) \end{gathered}$$

We write

$$\begin{aligned} \alpha &= \alpha_0(\vec{d}) \text{ where } \{d_1 = \alpha_1(\vec{d}), \dots, d_n = \alpha_n(\vec{d})\} \\ \overline{\alpha} &= \alpha_0(\vec{d}) \,\overline{\text{where}}\, \{d_1 = \alpha_1(\vec{d}), \dots, d_n = \alpha_n(\vec{d})\} \end{aligned}$$

• Recursors "code" algorithms up to (the obvious notion of) **recursor isomorphism**.

• Recursors are infinitary objects, but some "finiteness" restriction is built in the structure, by allowing only finitely many "parts".

Meaningful semantics for $\boldsymbol{\lambda}_r(\vec{c})$

$$(A, \pi) \mapsto \mathsf{rint}(A, \pi) \mapsto \overline{\mathsf{rint}(A, \pi)} = \mathsf{den}(A, \pi)$$

- The **referential intension** $\mathsf{rint}(A, \pi)$ is a recursor, modeling "the natural algorithm" which attempts to compute $\mathsf{den}(A, \pi)$.

The synonymy relation becomes:

$$A =_i B \iff (\forall \pi)[\mathsf{rint}(A, \pi) \simeq \mathsf{rint}(B, \pi)]$$

where $\simeq$ stands for recursor isomorphism.

Basic results

- The synonymy relation is decidable.

(NP and no better than graph isomorphism)

- We can add to the language a binary propositional attitude

$$\mathsf{SYN}[A, B] \iff A =_i B$$

and synonymy remains decidable in $\boldsymbol{\lambda}_r(\vec{c}, \mathsf{SYN})$.

Does all this help?

- Satisfactory sense-semantics for simple fragments (with no propositional attitudes)

'John loves Mary' $=_i$ 'Mary is loved by John'

'John and Mary love bicycling'
$=_i$ 'Mary and John love bicycling'

- Examples of propositional attitudes whose "meaning" we know, e.g.,

$$\mathsf{NEG}[A] \iff A \text{ is negative}$$

- A precise conjecture on the kind of operations that propositional attitudes (knowledge, belief) should be:

roughly: they should be operators on recursors which are "uniformly definable" in $\boldsymbol{\lambda}_r(\vec{c})$.