# A LOGICAL CALCULUS OF MEANING AND SYNONYMY

YIANNIS N. MOSCHOVAKIS

In his development of formal semantics for natural language Montague [1970a]–Montague [1973],[1] Richard Montague modeled the *meaning* (Frege's sense) of a term $A$ by its *Carnap intension* $\mathrm{CI}(A)$, the function which assigns to each *state* $a$, specifying a "possible world", "time" and "context of use", the *denotation* $\mathrm{den}(A)(a)$ of $A$ in that state. Now this is surely not right: among other things, it makes "there are infinitely many odd numbers" synonymous with "there are infinitely many prime numbers", which it is not.[2] At the other extreme, "structural" approaches to the modeling of meaning (like Russell's propositions,[3] Church [1946]–Church [1974] and Cresswell [1985]) basically tell us no more than that "the sense of a complex term $A$ can be determined from the syntactic structure of $A$ and the senses or denotations of the basic constituent parts of $A$", without explaining how this "determination" is to take place. But, to oversimplify Davidson's eloquent criticism in Davidson [1967], Theaetetus and the property of flying do not (by themselves) amount to the meaning of "Theaetetus flies": we would like to know just what kind of objects meanings are, and how the meaning of "Theaetetus flies" is determined by the meanings of 'Theaetetus' and 'flying'.

In Moschovakis [1994] I argued that the meaning of a term $A$ can be faithfully modeled by its *referential intension* $\mathrm{int}(A)$, an (abstract, idealized, not necessarily implementable) algorithm which computes the denotation of $A$. The basic technical tool in that paper was the *Formal language of recursion* FLR, for which the theory of referential intensions can be developed rigorously, and the applications to fragments of natural language were to come by "formalizing" (translating, rendering) them into FLR. Here I will develop the theory of referential intensions for the formal language $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, which extends the typed $\lambda$-calculus and so can accommodate (via the work of Montague) reasonably large fragments of natural language.

The claim that *meanings are algorithms* is a philosophical one, but this is primarily a paper in logic, not in the philosophy of language or in linguistics. Every discussion

The paper developed from a set of notes produced for a short course in NASSLLI '03.

I am grateful to Uwe Mönnich, Larry Moss, Isidora Stojanovic and the anonymous referees for many useful suggestions, and (especially) Fritz Hamm and my student Eleni Kalyvianaki, for innumerable conversations on the topic of this paper and invaluable criticism of early drafts. Kalyvianaki's most important contributions are invisible: she found errors and corrected some of the statements and (especially) the proofs which have been omitted from this paper.

Some of the missing proofs are posted on `http://www.math.ucla.edu/~ynm/papers.htm`.

[1] These have all been reprinted in Montague [1974], which also contains additional, relevant articles.

[2] A distinction between "meaning" and "sense" is introduced in Montague [1970c][Section 4] to allow a de re evaluation of demonstratives, but this does not affect mathematical assertions.

[3] See Pelham and Urquhart [1994].

of a natural language example will start with an assertion of the form

(1) every man loves some woman

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

which will be explained and motivated when not obvious, but cannot be rigorously justified, as I will not specify with any precision the all-important *rendering* (or *translation*) operation $\ldots \xrightarrow{\text{render}} A$.[4] Some would argue that this is the most important part of the extraction of meanings from linguistic expressions, and I would agree with them. On the other hand, I think that the theory of what-happens-next proposed here may be of some value, primarily because of two reasons.

First, the modeling of meanings by referential intensions goes far beyond the imagery and analogy with computation often used to explain the relation between Frege's sense and denotation, especially by Dummett.[5] The explication of *meaning* by *abstract algorithm* is analogous to the "definition" of ordered pairs in axiomatic set theory: necessarily complex and somewhat forbidding at first sight, it codifies the structural properties of a specific understanding of meaning which (with some effort) can be understood intuitively and used for direct philosophical and linguistic analysis independent of the technicalities.

Second, the formal processing of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$-terms (the "calculus" of the title) sets conditions and limitations on the rendering operation, it provides new ways to implement some syntactic transformations which affect meaning (like co-indexing and co-ordination), and for some English phrases, it suggests some plausible, *novel renderings directly in* $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ which are not referentially synonymous with any terms of the typed $\lambda$-calculus. The examples in §3.24 – §3.27 are admittedly very simple, but they employ general methods which may prove useful in computational semantics.

The technical part of the paper—the logic—is found mostly in Sections 1 and 3. It is illustrated and motivated by a number of simple examples in the middle Section 2, and by a discussion of two relevant, standard puzzles about meanings in Section 4. The last, pretentiously titled Section 5 reviews and motivates briefly the connection between algorithms and meanings, and the specific, mathematical notion of abstract algorithm which I use to model meanings.

---

[4] In fact the full rendering operation is of the form

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

where the (informally understood) *context* determines not only the state (as we will make it precise in Subsection §2.2), but also which precise reading of the expression is appropriate and what formal transformations should be made (e.g., co-indexing), depending on information about "what the speaker meant", intonation, if the expression was spoken, punctuation and capitalization, if it was written, etc. I will have nothing to say about these factors and how they determine the formal state, and so I will concentrate on the simpler, syntactical component of rendering

$$\text{natural language expression} \xrightarrow{\text{render}} \text{formal expression}$$

for which the subsequent extraction of meaning will provide some suggestions.

[5] Cf. the discussion and the references to Dummett [1978] and Evans [1982] in the introduction to Moschovakis [1994].

| | | | |
|---|---|---|---|
| Names of "pure" objects | $0, 1, 2, \emptyset, \ldots$ | : | $\mathsf{e}$ |
| Names, demonstratives | John, I, he, him, today | : | $\tilde{\mathsf{e}}$ |
| Common nouns | man, unicorn, temperature | : | $\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ |
| Adjectives | tall, young | : | $(\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}})$ |
| Propositions | it rains | : | $\tilde{\mathsf{t}}$ |
| Intransitive verbs | stand, run, rise | : | $\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ |
| Transitive verbs | find, loves, be | : | $\tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ |
| Adverbs | rapidly | : | $(\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}})$ |

TABLE 1. Empirical constants.

In lectures on this topic, I sometimes use the subtitle "*derived by taking programming languages seriously*", to emphasize the dependence of this theory on algorithmic ideas, which goes much deeper than the modeling of meanings by abstract algorithms.

**§1. The typed $\lambda$-calculus with acyclic recursion, $\mathsf{L}^{\lambda}_{\mathbf{ar}}$.** The language $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ is a *typed calculus of terms*, an extension of the *two-sorted type theory* $\mathrm{Ty}_2$ of Gallin [1975][§8] into which the language of *intensional logic* LIL of Montague [1973] can be interpreted by Gallin's Theorem 8.2.[6] Each term $A$ of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ is tagged with a type $\sigma$, and (for each assignment g to its free variables) denotes an object $\mathrm{den}(A)(\mathrm{g})$ in some specified universe $\mathbb{T}_\sigma$ of *objects of type* $\sigma$.[7] We will give a brief description of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, following closely Gallin [1975] and concentrating on the two substantial ways in which $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ extends $\mathrm{Ty}_2$.

§1.1. **Syntax**. This is determined by the *types*, the *constants*, the *variables* and the *terms*.

**Types** are defined recursively, starting with the basic types $\mathsf{e}$ of *entities*, $\mathsf{t}$ of *truth values*, and $\mathsf{s}$ of *states*, and allowing the formation of arbitrary *function types* $(\sigma \to \tau)$. In the shorthand used for simple recursive definitions by computer scientists,[8]

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

[6]Gallin is concerned only with showing that each term of LIL is denotationally equal (in a suitable sense) with a term of $\mathrm{Ty}_2$, and does not worry about meanings. His interpretation can be combined with the methods of this paper to assign natural, robust meanings to the terms of LIL, but this requires a careful comparison of LIL with $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ and we will leave it for the forthcoming Kalyvianaki and Moschovakis [].

[7]LIL does not have a symbol $\mathsf{s}$ for the type of "state" or variables which range over states, and each term $A$ of type $\sigma$ defines a function $\mathrm{CI}(A) : \mathbb{T}_{\mathsf{s}} \to \mathbb{T}_\sigma$ from the states to the objects of type $\sigma$. Perhaps Montague made these choices because we cannot explicitly refer to the state in natural language, although like every formal language, LIL has terms which do not render anything we could utter, including free variables (of any type). The lack of state variables causes considerable awkwardness in the development of logic, and the (effective) absence of pure types interferes with the development of a satisfactory theory of meaning for mathematical statements.

[8]I use "$\equiv$" for the identity relation between the syntactic objects of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ (types and terms), to avoid confusion with "=", which is itself a syntactic object denoting the identity relation between the objects that $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ is about.

i.e., the set of types is the smallest set which includes the distinct "symbols" $\mathsf{e}, \mathsf{t}, \mathsf{s}$ and is closed under the pairing operation $(\sigma_1 \to \sigma_2)$. A type is *pure* (or *state-free*) if the state type $\mathsf{s}$ does not occur in it:

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

Especially significant for the intended interpretations are the types $(\mathsf{s} \to \sigma)$ of "state-dependent" objects: in particular, we let

(2) $$\tilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t}) \equiv \text{the type of Carnap intensions,}$$

(3) $$\tilde{\mathsf{e}} \equiv (\mathsf{s} \to \mathsf{e}) \equiv \text{the type of Carnap individual concepts.}$$

It is also useful to introduce the notation

$$\tilde{\mathsf{q}} \equiv (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{t}} \equiv \text{the type of (state-dependent) unary quantifiers,}$$

which will type terms like "every woman", "some man", etc.

As is usual in the $\lambda$-calculus, we also set

$$\sigma_1 \times \sigma_2 \to \tau \equiv (\sigma_1 \to (\sigma_2 \to \tau)),$$

so that $\sigma_1 \times \sigma_2 \to \tau$ is the type of functions of two variables, which are "identified" with function-valued unary functions. This "currying" convention extends naturally to types and functions of three or more variables.

The types of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ specify kinds of semantic objects, and should not be confused with the syntactic categories of natural language. Many syntactic categories may be mapped onto the same type: for example, intransitive verbs $(\mathsf{run})$ and proper nouns $(\mathsf{man})$ are both rendered by terms of the same $\mathsf{L}^{\lambda}_{\mathrm{ar}}$-type $(\tilde{\mathsf{e}} \to \tilde{\mathsf{t}})$, although their syntactic categories are obviously distinct.

**Constants**. We assume given a (finite) set $K$ of typed constants, the "vocabulary", and we write $\mathsf{c} : \sigma$ to indicate that $\mathsf{c}$ has type $\sigma$. Examples of constants are given in Table 1, but we will add more later on—including the usual *logical constants* in Table 3. Notice the absence of *propositional attitudes* (believe that, assert that) with which we will not deal in this paper, except for some comments.

The syntactically correct terms depend on the choice of $K$, and so we write $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$ for the language determined from a specific $K$.

**Variables**. For each type $\sigma$, $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ has two infinite sequences of variables,

- the *pure variables* $\mathsf{v}^{\sigma}_0, \mathsf{v}^{\sigma}_1, \ldots$, and
- the *recursion variables* or *locations* $\mathsf{p}^{\sigma}_0, \mathsf{p}^{\sigma}_1, \ldots$

Syntactically, pure variables are *quantified*, while locations are *assigned-to*. The separation of these two roles of variables is essential in the development of the intensional semantics of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$.

**Terms** are defined recursively, starting with the variables and the constants and using *application*, *$\lambda$-abstraction* and (mutual) *acyclic recursion*, which we will explain in the next paragraph. The definition also assigns a type to every term and specifies the free and bound occurrences of variables in it. We write

$$A : \sigma \iff A \text{ has type } \sigma,$$

and in accordance with the currying convention discussed above, if $A : \sigma_1 \times \sigma_2 \to \tau$, $B : \sigma_1$ and $C : \sigma_2$, we write synonymously

$$A(B, C) \equiv A(B)(C).$$

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

$\mathsf{c}$ is a constant of type $\sigma$, and (as a term) $\mathsf{c} : \sigma$

$x$ is a variable of either kind, of type $\sigma$, and (as a term) $x : \sigma$

$C : \sigma$, $B : (\sigma \to \tau)$, and $B(C) : \tau$

$B : \tau$, $v$ is a pure variable of type $\sigma$, and $\lambda(v)(B) : (\sigma \to \tau)$

$n \geq 1$, $A_i : \sigma_i$, $p_1, \ldots, p_n$ are distinct locations, $p_i : \sigma_i$,
the system $\{p_1 := A_1, \ldots, p_n := A_n\}$ is acyclic,
and $A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\} : \sigma_0$

In addition (recursively) all occurrences of $v$ are bound in $\lambda(v)(B)$, and all occurrences of $p_1, \ldots, p_n$ are bound in $A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$; occurrences of variables not bound by this clause are free.

TABLE 2. The terms of $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$.

Table 2 summarizes the definition with all the necessary side conditions, except for *acyclicity*, which we define next.

§1.2. **Acyclic recursion**. The best, intuitive way to understand the recursive construct is to read "where" more-or-less normally:

$$\mathsf{loves}(j, s) \text{ where } \{j := \mathsf{John}, m := \mathsf{Mary}, s := \mathsf{sister}(m)\}$$

communicates the same information as

$$\text{if } j \text{ is John, } m \text{ is Mary, and } s \text{ is the sister of } m, \text{ then } j \text{ loves } s,$$

in other words "John loves Mary's sister"—and, as we will prove, this term is referentially synonymous with

$$\mathsf{loves}(\mathsf{John}, \mathsf{sister}(\mathsf{Mary}))$$

which renders "John loves Mary's sister".

Formally, a system of *assignments* $\{p_1 := A_1, \ldots, p_n := A_n\}$ is *acyclic* if it is possible to associate a natural number $\mathrm{rank}(p_i)$ with each of the locations, so that

$$\text{if } p_j \text{ occurs free in } A_i, \text{ then } \mathrm{rank}(p_i) > \mathrm{rank}(p_j);$$

the obvious idea is that $p_i$ has higher rank than $p_j$ if its value "depends" (or could depend) on that of $p_j$. For example, the system

$$\{f := \mathsf{father}(m), m := \mathsf{mother}(j), j := \mathsf{John}\} \text{ is acyclic,}$$

with $\mathrm{rank}(j) = 0, \mathrm{rank}(m) = 1, \mathrm{rank}(f) = 2$, while the one-assignment system

$$\{p := c(p)\} \text{ is not acyclic,}$$

because any ranking of $p$ would need to satisfy $\mathrm{rank}(p) > \mathrm{rank}(p)$, which it cannot. Acyclic systems express "trivial" systems of "recursive definitions" which "close off" (and produce unique values) in a finite number of steps.[9]

§1.3. **Congruence**. Two terms are *congruent* if one can be obtained from the other by alphabetic changes of the bound variables and re-orderings of the assignments within the acyclic recursion construct. Formally, *congruence* is the smallest equivalence relation $\equiv_c$ between terms which respects alphabetic replacement of bound variables (of both kinds), application, $\lambda$-abstraction and acyclic recursion, and such that for any permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$,

$$A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \equiv_c A_0 \text{ where } \{p_{\pi(1)} := A_{\pi(1)}, \ldots, p_{\pi(n)} := A_{\pi(n)}\},$$

so that, for example,

$$A \text{ where } \{p := B, q := C\} \equiv_c A \text{ where } \{q := C, p := B\}.$$

The last condition means that the assignments within { } are interpreted as a *set*, not a sequence. Both the denotational and intensional semantics of $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$ will respect congruence, and so we will sometimes tacitly identify congruent terms.

A term $A$ is *explicit* if the constant where does not occur in it; *recursive* if it is of the form $A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$; and a *$\lambda$-calculus term* if it is explicit and no recursion variable occurs in it, i.e., if it is a term of Gallin's $\mathrm{Ty}_2$. A term is *closed* if it has no free occurrences of variables.

As is usual, we will "misspell" types and terms, skipping parentheses and font-changes, inserting brackets to indicate grouping, writing "$A$ loves $B$" and "$A = B$" rather than "loves$(A)(B)$" and "$= (A)(B)$", etc. It will also be useful to allow on occasion the *dummy recursion construct* $A$ where { } as a misspelling of $A$, i.e.,

$$A \text{ where } \{\ \} \equiv_{\mathrm{def}} A. \tag{4}$$

§1.4. **Denotational semantics**. $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$ is interpreted in structures of the form

$$\mathfrak{A} = (\{\mathbb{T}_\sigma\}_{\sigma \in \mathrm{Types}}, \{c\}_{\mathsf{c} \in K}, \mathrm{den})$$

satisfying the following conditions (S1) – (S4):

(S1) *Each $\mathbb{T}_\sigma$ is a set*. We further assume that the set of *entities* $\mathbb{T}_{\mathsf{e}}$ contains (at least) three distinct objects $0, 1, er$; that the set of *states* $\mathbb{T}_{\mathsf{s}}$ is not empty; and (for convenience) that the set of *truth values* $\mathbb{T}_{\mathsf{t}}$ coincides with the set of entities,

$$\mathbb{T}_{\mathsf{t}} = \mathbb{T}_{\mathsf{e}}.$$

(S2) *Each $p \in \mathbb{T}_{(\sigma\to\tau)}$ is a function $p : \mathbb{T}_\sigma \to \mathbb{T}_\tau$.*

(S3) *If $\mathsf{c} : \sigma$, then $c \in \mathbb{T}_\sigma$.*

[9] If we remove the acyclicity restriction on the recursion construct, we obtain the *$\lambda$-calculus with full recursion* $\mathsf{L}^{\lambda}_{\mathrm{r}}$, a mild extension of the language *PCF* which has been extensively studied by computer scientists, cf. Plotkin [1977]. Essentially all the results of this paper can be extended to $\mathsf{L}^{\lambda}_{\mathrm{r}}$, but at a heavy price in mathematical technicalities, starting with the need to develop different (and substantially more complex) denotational semantics. Full recursion is admitted in FLR, and some applications of it to the philosophical analysis of self-reference were included in Moschovakis [1994]; it is a moot point whether its extension to $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ can contribute enough to computational semantics to be worth the considerable extra work.

(S4) den *is a function which associates with each term* $A : \sigma$ *and each assignment* g *to the variables an object* $\mathrm{den}(A)(\mathrm{g}) \in \mathbb{T}_\sigma$ *so that the following conditions hold*:[10]

(D1) $\mathrm{den}(x)(\mathrm{g}) = \mathrm{g}(x)$; $\mathrm{den}(\mathsf{c})(\mathrm{g}) = c$.

(D2) $\mathrm{den}(A(B))(\mathrm{g}) = \mathrm{den}(A)(\mathrm{g})(\mathrm{den}(B)(\mathrm{g}))$.

(D3) $\mathrm{den}(\lambda(v)(B))(\mathrm{g}) = h$, where, for all $t$, $h(t) = \mathrm{den}(B)(\mathrm{g}\{v := t\})$.

(D4) $\mathrm{den}(A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\})(\mathrm{g})$
$= \mathrm{den}(A_0)(\mathrm{g}\{p_1 := \overline{p}_1, \ldots, p_n := \overline{p}_n\})$,
where the values $\overline{p}_i$ are defined for $i = 1, \ldots, n$ by recursion on $\mathrm{rank}(p_i)$:
$$\overline{p}_i = \mathrm{den}(A_i)(\mathrm{g}\{p_{k_1} := \overline{p}_{k_1}, \ldots, p_{k_m} := \overline{p}_{k_m}\}),$$
where $p_{k_1}, \ldots, p_{k_m}$ are the variables with ranks lower than $\mathrm{rank}(p_i)$.

It is easy to check (by induction on the terms) that at most one denotation function satisfying (S4) exists (for any given choice of $\mathbb{T}_\sigma$ so that (S1) – (S3) holds), and that $\mathrm{den}(A)(\mathrm{g})$ depends only on the values $\mathrm{g}(x)$ for those variables which have free occurrences in $A$. For denotational term equality we will use the familiar notation from logic,

$$\models A = B \iff \text{for all assignments g}, \mathrm{den}(A)(\mathrm{g}) = \mathrm{den}(B)(\mathrm{g}). \tag{5}$$

The structure $\mathfrak{A}$ is *standard* if for all $\sigma, \tau$,
$$\mathbb{T}_{(\sigma\to\tau)} = \text{the set of all functions } p : \mathbb{T}_\sigma \to \mathbb{T}_\tau.$$
A standard $\mathfrak{A}$ is determined by the basic sets $\mathbb{T}_{\mathsf{e}}$, $\mathbb{T}_{\mathsf{s}}$ and the interpretations $\mathsf{c} \mapsto c$ of the constants, as (S4) can then be viewed as a recursive definition of the (unique) denotation function.

To illustrate the computation of denotations in the recursive case, consider the closed term

$$A \equiv p \,\&\, q \text{ where } \{p := \mathsf{loves}(j, m), q := \mathsf{dislikes}(j, h), \\ h := \mathsf{husband}(m), j := \mathsf{John}, m := \mathsf{Mary}\}. \tag{6}$$

Assuming that the indicated constants name the obvious objects and relations, we compute the denotation of $A$ in stages, as follows:

Stage 1: $j :=$ John, $m :=$ Mary
Stage 2: $h := \mathrm{husband}(m) =$ Mary's husband
$p := \mathrm{loves}(j, m) =$ the truth value of "John loves Mary"
Stage 3: $q := \mathrm{dislikes}(j, h)$
$=$ the truth value of "John dislikes Mary's husband"
Stage 4: $\mathrm{den}(A) = p \,\&\, q$

so that, finally,

$\mathrm{den}(A) =$ the truth value of "John loves Mary and he dislikes her husband".

[10] An *assignment* (or *valuation*) is a function g which assigns to each pure or recursion variable $x$ of type $\sigma$ an object $\mathrm{g}(x) \in \mathbb{T}_\sigma$. If $x : \sigma$ is a variable and $t \in \mathbb{T}_\sigma$, then the *update* of g by the assignment $x := t$ is defined in the obvious way,
$$\mathrm{g}\{x := t\}(y) = \begin{cases} t, & \text{if } y \equiv x, \\ \mathrm{g}(y), & \text{otherwise.} \end{cases}$$

§1.5. **The intended interpretation**. To interpret natural language in $\mathfrak{A}$, we assume that $\mathbb{T}_{\mathsf{e}}$ contains the natural numbers $\mathbb{N} = \{0, 1, \dots\}$, the real numbers and other mathematical objects, but also people (dead or alive, or who might live in some alternate universe), trees, points in spacetime, etc.[11]

A state $a$ (intuitively) specifies a "full context" in which the terms of $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$ can be interpreted. We will say more about states in Section 2, but, for the technical definitions in this section, all we need is that $\mathbb{T}_{\mathsf{s}}$ is some non-empty set. The pure objects are built up from the members of $\mathbb{T}_{\mathsf{e}}$ and do not depend on the choice of $\mathbb{T}_{\mathsf{s}}$.

Finally, the identification $\mathbb{T}_{\mathsf{e}} = \mathbb{T}_{\mathsf{t}}$ sounds a bit peculiar, but it is both economical and useful: we identify "truth" with the number 1, "falsity" with the number 0, and we assign *er* (*error*) to terms of type $\mathsf{e}$ or $\mathsf{t}$ which have no natural truth value. The correct interpretations of the constants should assign *er* to both "Mary's husband" and "Mary's husband is tall", if Mary is not married or has more than one husband.

It is sometimes not clear (or a matter of choice) whether a phrase should be rended by a constant or a term. For example, the language might have a constant $\mathsf{husband}$, or render "Mary's husband" with the term $\mathsf{the}(\lambda(x)\mathsf{married}(x, \mathsf{Mary}))$. We will make such choices explicit, when we need to be specific, without taking a position on which is "the right choice".

Whether the intended $\mathfrak{A}$ is standard or not is a philosophical question: some might admit in the model only those individual concepts and Carnap intensions which are definable. One of the methodological principles which underlies this work is that *logic should provide a framework for philosophical inquiry and linguistic analysis, but should not decide between coherent philosophical alternatives or plausible formalizations of natural language phrases*. So for the remainder of this paper, *we fix one* (possibly non-standard) *structure*, without placing on it any restrictions beyond (S1) – (S4).

§1.6. **Formal replacement and the replacement property**. As usual, if $A$ is a term, $x$ is a variable or a constant of type $\sigma$ and $C$ is a term of type $\sigma$, then

$$A\{x :\equiv C\} \equiv \text{the result of replacing } x \text{ by } C \text{ in all its free occurrences in } A,$$

where, of course, all occurrences of a constant are free. The replacement is *free* if no free occurrence of a variable in $C$ becomes bound in $A\{x :\equiv C\}$, and in this case, easily, for every assignment g, if $x$ is a variable

$$\mathrm{den}(A\{x :\equiv C\})(\mathrm{g}) = \mathrm{den}(A)(\mathrm{g}\{x := \mathrm{den}(C)(\mathrm{g})\}),$$

and if $x \equiv \mathsf{c}$ is a constant, then

$$\text{if } \mathrm{den}(\mathsf{c}) = \mathrm{den}(C)(\mathrm{g}), \text{ then } \mathrm{den}(A\{\mathsf{c} :\equiv C\})(\mathrm{g}) = \mathrm{den}(A)(\mathrm{g}).$$

We will tacitly assume that all replacements are free when we apply this operation.

§1.7. $\mathsf{L}^{\lambda}_{\mathbf{ar}}$ **vs. the typed $\lambda$-calculus**. It can be easily shown that *every term with no free locations is denotationally equal to an explicit term*, so that, as far as denotations go, there is no need for the acyclic recursion construct. On the other hand, we will show that $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ *is intensionally more expressive than Gallin's* $\mathrm{Ty}_2$, for example the term $A$ in (6) is not referentially synonymous with any explicit term (so long as the constants which occur in it are not given truly perverse interpretations).

[11] If we put all sets in $\mathbb{T}_{\mathsf{e}}$, as we should, then it is a proper class and not a set. I will disregard this technical wrinkle, which can be easily corrected by introducing some irrelevant technicalities.

**§2. Examples.** Beyond fleshing out some of the formal definitions of Section 1, the simple examples in this section will help illustrate the modeling of meaning coming up next. To facilitate discussing meanings in the examples, let us jump the gun and introduce here the notation

(7) $$A \approx B \iff A \text{ and } B \text{ are referentially synonymous.}$$

The precise definition is given in §3.21.

First a cautionary note.

§2.1. **$\beta$-conversion**. This is the most basic rule of the $\lambda$-calculus: *if the variable $u$ and the term $B$ have the same type, and if the substitution is free, then*

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

For example,

$$\models \Big(\lambda(j)\mathsf{loves}(j,j)\Big)(\mathsf{John}) = \mathsf{loves}(\mathsf{John},\mathsf{John}).$$

The rule of $\beta$-conversion does not hold for referential synonymy, in fact

$$\Big(\lambda(j)\mathsf{loves}(j,j)\Big)(\mathsf{John}) \not\approx \mathsf{loves}(\mathsf{John},\mathsf{John});$$

this is not unexpected, because these terms render English sentences which are not usually perceived as synonymous,[12]

(JLH) $$\text{John loves himself} \xrightarrow{\text{render}} \Big(\lambda(j)\mathsf{loves}(j,j)\Big)(\mathsf{John}),$$

(JLJ) $$\text{John loves John} \xrightarrow{\text{render}} \mathsf{loves}(\mathsf{John},\mathsf{John}).$$

In fact, *$\beta$-conversion almost never preserves meaning*, just as logical deduction does not—otherwise all theorems would be synonymous, which is absurd; so it is important not to assume it unthinkingly in analyzing the examples.

§2.2. **States**. To be specific, we will assume in this paper that a *state* is a quadruple

$$a = \langle i, j, k, A, \delta\rangle$$

which specifies a *possible world* $i$, a *moment of time* $j$, a *point in space* $k$, a speaker (or "agent") $A$, and a function $\delta$ which assigns values to all possible occurrences of proper names and demonstratives, indexed by the order in which they appear in terms: so it might be that

$$\delta(\mathsf{John}_1) = \text{John Steinbeck},\ \delta(\mathsf{John}_2) = \text{John Wayne}, \dots,$$
$$\delta(\mathsf{I}_1) = \text{Yiannis Moschovakis}, \dots, \delta(\mathsf{today}_1) = \text{August 4, 2004} \dots$$

For example, to determine the truth value of

"John loves her and she loves him",

[12]One can argue that to understand (JLH) you need to know how to render himself, which you do not need to understand (JLJ); and from the computational point of view, that to compute the truth value of (JLJ) you need to compute the reference of John twice, while to decide (JLH) you need compute that reference only once.

we must know when the sentence was asserted (because love fades), but also who "John", "her", "she", "him" are—and there could be as few as two and as many as four persons involved.

We will refer to the values specified by a state as

$$\text{world}(a),\ \text{time}(a),\ \text{agent}(a),\ \text{John}_1(a),\ \text{I}_1(a),\ \text{he}_2(a),\ \text{etc},$$

and *we will leave open the question of which states exist* in the basic set $\mathbb{T}_{\mathsf{s}}$ of the intended interpretation $\mathfrak{A}$, in accordance with the discussion in §1.5. The choice of $\mathbb{T}_{\mathsf{s}}$ does not affect the way in which the denotations and referential intensions of terms in $\mathfrak{A}$ are computed; but it does, of course, determine their values, and so, to explore the examples, we will sometimes make some innocuous assumptions—e.g., that it may rain in some states while it is sunny in others, so that, in particular, there are at least two states. Such assumptions will be natural and non-controversial: we will not need to consider whether there are states in which $\text{agent}(a) \neq \text{I}(a)$, or $\text{today}_1(a) \neq \text{today}_2(a)$ which touch on difficult questions of philosophy and the expressibility of natural language.

§2.3. **Carnap objects of type $(\mathsf{s} \to \sigma)$; rigidity**. These include the denotations of proper names like John and the demonstratives He, her, today, ... , of type $\tilde{\mathsf{e}}$, as well as the *Carnap intensions* of type $\tilde{\mathsf{t}}$, for example it rains, for which

$$\text{it rains}(a) = 1 \iff \text{it is raining in the state } a.$$

A Carnap object $x : (\mathsf{s} \to \sigma)$ is *rigid* if, for all states $a, b$, $x(a) = x(b)$; and a closed term $A : (\mathsf{s} \to \sigma)$ is rigid if it denotes a rigid object, i.e., a constant function $p : (\mathsf{s} \to \sigma)$ such that $p(a) = y$ for some fixed $y : \sigma$.

If we set[13]

$$\text{dere}(x, a)(b) = x(a) \quad (x : \mathsf{s} \to \sigma, a, b \in \mathbb{T}_{\mathsf{s}}) \tag{8}$$

then the object $\text{dere}(x, a) : (\mathsf{s} \to \sigma)$ is rigid and denotes $x(a)$ in every state. It is quite standard in philosophy of language today to assume that at least some historical proper names ("Aristotle") are rigid, but we will neither assume nor forbid this here.

§2.4. **Connectives, quantifiers and the identity relation**. The "pure" operations of logic which we use to express mathematical sentences are objects of pure type, e.g.,

$$\neg : \mathsf{t} \to \mathsf{t}, \quad \& : \mathsf{t} \times \mathsf{t} \to \mathsf{t}, \ldots$$

and they are defined as usual on $\mathbb{T}_{\mathsf{t}} = \mathbb{T}_e$, with 1 and 0 standing for truth and falsity and the obvious treatment of errors: for example,

$$\neg(t) = \begin{cases} 1 - t, & \text{if } t \in \{0, 1\}, \\ er, & \text{otherwise} \end{cases}, \qquad \&(s, t) = \begin{cases} \min(s, t), & \text{if } s, t \in \{0, 1\}, \\ er, & \text{otherwise} \end{cases}$$

[13]This is really Kaplan's dthat$(x, a)$ in Kaplan [1978a], but I am using a different notation to avoid confusion, since Kaplan's understanding (and use) of this important construct is somewhat different from the present one. The notation dere$(x, a)$ comes from the use we will make of these functions later on to distinguish between "de re" and "de dicto" readings of terms in modal contexts. I am not assuming that the language has a constant dere—I wouldn't know the English word for it.

etc. More interesting for the natural language examples are the state-depended versions of these operations, for which we use the same notations: the types of these are specified in Table 3 and their definitions are again obvious, e.g.,

$$\neg(p,a) = \neg(p(a)), \quad =(p,q,a) = \begin{cases} 1, & \text{if } p(a) = q(a) \neq er, \\ 0, & \text{if } p(a), q(a) \in \mathbb{T}_{\mathsf{e}} \text{ and } p(a) \neq q(a), \\ er, & \text{otherwise, i.e., if } p(a) = er \text{ or } q(a) = er, \end{cases}$$

and leaving out the treatment of errors (as we will always do in the sequel),

$$\text{every}(p)(q)(a) \iff (\forall x : (\mathsf{s} \to \mathsf{e}))[\neg p(x,a) \vee q(x,a)].$$

For a (pedantically spelled out) example of term-formation with these constants, consider (1).

§2.5. **Modal operators**. We assume the language has a constant $\Box$ for the basic necessity operator, Montague's "full necessity", or "necessarily always", as Thomason calls it:

$$\Box(p)(a) \iff (\forall b)p(b).$$

Kaplan [1978b] argues convincingly that this interpretation is inappropriate for terms which contain demonstratives, but in our determination to avoid philosophical commitments, it is best to allow his interpretation as a de re reading of the modality, without forbidding the de dicto reading. With the notation

$$\Box_2(p,x,y) \iff x \text{ and } y \text{ necessarily have property } p \quad (p : \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}, x, y : \tilde{\mathsf{e}})$$

for binary properties (and $\Box_n$ for $n$-ary ones), the correct definition is

$$\Box_2(p,x,y)(a) = \Box(p(\text{dere}(x,a), \text{dere}(y,a)))(a).$$

For example, with a primitive $\mathsf{reside} : \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ for "$x$ is in place $y$", this allows four readings for "I am necessarily here",

$$\Box(\mathsf{reside}(\mathsf{I}, \mathsf{here})),\ \Box_1(\lambda(x)\mathsf{reside}(x, \mathsf{here}), \mathsf{I}),$$
$$\Box_1(\lambda(y)\mathsf{reside}(\mathsf{I}, y), \mathsf{here}),\ \Box_2(\mathsf{reside}, \mathsf{I}, \mathsf{here}),$$

of which the first conveys the information that "the speaker is necessarily at the place the utterance is made" and the last one claims that "Moschovakis is necessarily in Phaliron, Greece" if I were to say something as I write this; Kaplan would disallow all but the last, which is, of course, false.

The technique works for any modal operator: for the unary case, if $F : (\tilde{\mathsf{t}} \to \tilde{\mathsf{t}})$, then

$$F_1(p,x)(a) = F(p(\text{dere}(x,a)))(a)$$

produces the corresponding de re version, with type $(\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$.

For serious work on the modal part of the language, we would obviously need to introduce additional modal constants, in the past, in all possible worlds (but at the present time and location), etc.

| | |
|---|---|
| $=$ | $: \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ |
| $\neg, \Box,$ in the future | $: \tilde{\mathsf{t}} \to \tilde{\mathsf{t}}$ |
| $\&, \vee$ | $: \tilde{\mathsf{t}} \times \tilde{\mathsf{t}} \to \tilde{\mathsf{t}}$ |
| every, some | $: (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{q}}$ |
| the | $: (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{e}}$ |

TABLE 3. Logical constants.

§2.6. **Descriptions**. The natural definition of the description operator returns an error if the existence and uniqueness conditions are not fulfilled:

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

where $b \mapsto y$ is the constant function on the states with value $y$. Notice that we do not ask for a unique $x \in \mathbb{T}_{\tilde{\mathsf{e}}}$, but only for a unique $y \in \mathbb{T}_{\mathsf{e}}$ which satisfies the relevant condition in the relevant state $a$. Thus, assuming that "$x$ is married to $y$" is unambiguously determined in each state, we can set

$$\text{Mary's husband} \xrightarrow{\text{render}} \mathsf{the}(\lambda(x)\mathsf{married}(x, \mathsf{Mary})),$$

and this will give us the correct value in every state, no matter how often Mary gets married. Moreover,[14]

$$\text{Mary's husband is tall} \xrightarrow{\text{render}} \mathsf{tall}(\mathsf{man})(\mathsf{Mary's\ husband}),$$

and this term automatically gets the right value in every state, including *er* in a state in which Mary does not have a unique husband, on the assumption that $\text{tall}(er) = er$. And "the King of France is bald" will also be assigned *er* today, contrary to Russell's wishes.[15]

§2.7. **Locality and modality**. An object

$$p : (\mathsf{s} \to \sigma) \to \tilde{\mathsf{t}}$$

is *local*[16] if each value $p(x, a)$ depends only on $x(a)$ and not on any other values $x(b)$, i.e., if

$$\text{for all } x, x', a, \text{ if } x(a) = x'(a), \text{ then } p(x, a) = p(x', a),$$

or, equivalently,

$$\text{for all } x \text{ and } a,\ p(x, a) = p(\text{dere}(x, a), a).$$

[14]The type $(\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}})$ we have assigned to adjectives requires a noun as the argument of tall, and for this we must depend on the informal context of Footnote 4; it is assumed here that Mary's husband is classified as tall among men and not (for example) among basketball players.

[15]Cf. the discussion in Moschovakis [1994].

[16]See Montague [1973][Section 4]. Montague and Gallin use *extensional* and *intensional* for our *local* and *modal*, but this adds one more use to the already overloaded extension-intension distinction and suggests a connection between modality and meaning which is not in the spirit of this article.

An object which is not local is *modal*. A closed term $A : (\mathsf{s} \to \sigma) \to \tilde{\mathsf{t}}$ is local or modal accordingly as it denotes a local or modal object.

For example, the negation operation $\neg : (\tilde{\mathsf{t}} \to \tilde{\mathsf{t}})$ is local, while $\Box : (\tilde{\mathsf{t}} \to \tilde{\mathsf{t}})$ is modal, by their definitions above. What seems (at first) surprising is that some common nouns and verbs are also modal, in this abstract sense, and that the distinction is worth noting.

If, for example,

$$\text{The president is running} \xrightarrow{\text{render}} \mathsf{runs}(\mathsf{the}(\mathsf{president})),$$

then the constant $\mathsf{runs} : (\tilde{e} \to \tilde{\mathsf{t}})$ is most likely local, interpreted by the relation

$$\text{runs}(x, a) \iff x(a) \text{ is running in state } a.$$

However, in Partee's classic example

$$\text{the temperature is rising} \xrightarrow{\text{render}} \mathsf{rises}(\mathsf{the}(\mathsf{temperature})),$$

if $\mathsf{temperature} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ is a (local) constant defined by

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

then rises cannot be reasonably interpreted by a local object: because we cannot tell whether the temperature is rising in state $a$ from the mere knowledge of its value in $a$. We can interpret rises closest to our intuitions (and get the right truth value in the example) by setting

$$a\{j := t\} = \text{the state which differs from } a \text{ only in that time}(a\{j := t\}) = t,$$
$$\text{rises}(x, a) \iff \text{the function } t \mapsto x(a\{j := t\}) \text{ is increasing at time}(a),$$

or, more precisely (with a bit of calculus),[17]

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

This object "rises" is then modal.

For another example, consider the sentence

the color of the sky ranged from light pink to deep, brooding red;

the verb "ranges" is modal in this usage since to determine whether ranges(color, $a$) we must evaluate color($b$) for various states $b$ which differ in "observed location" from the current state $a$—assuming, for the example, that "observed location" is part of the state.

It is important to distinguish modality—which has to do with the dependence of $p(x, a)$ on values $x(b)$ for states $b \neq a$—from the possible dependence of $p(x, a)$ on properties of $x(a)$ in states $b \neq a$. To interpret the verb "runs", for example, we might use one of the following two plausible interpretations:

$$\text{runs}(x, a) \iff x(a) \text{ is running in state } a, \tag{9}$$
$$\text{runs-alt}(x, a) \iff x \text{ is running in state } a, \tag{10}$$

[17]This assumes that $x(a\{j := t\})$ is a real number for $t$ near time($a$). If not, then rises($x, a$) should probably be set to *er*.

where runs-alt$(x, a)$ is defined like rise$(x, a)$, using values of $x(a\{j := t\})$ for various $t$'s. Suppose that in the current state $a$,

$$\text{the}(\text{President})(a) = \text{Bush}.$$

With the (more natural, I think) first (local) interpretation, we cannot tell if

$$\text{runs}(\text{the}(\text{President}), a) \tag{11}$$

solely from a snapshot of Bush in the current state, as he may just be standing in a running posture; to assert (11) we need to observe Bush for a small time-interval around the current time—but we only need to observe Bush, not the person who might have been "the President" a few minutes before. Thus "runs" is local (directly from its definition), even though the truth value of runs$(x, a)$ may depend on properties of $x(a)$ in states $b$ other than $a$. On the other hand, by its definition again, the truth value of

$$\text{runs-alt}(\text{the President}, a)$$

depends on who "the President" is in states other than $a$, and it may be in some doubt right about inauguration time; this is a modal verb.[18]

These considerations extend directly to transitive verbs: an object $p : \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ is *local* if each value $p(x, y, a)$ depends only on $x(a)$ and $y(a)$, and there are obvious, natural notions of "local in the first variable" and "local in the second variable".

§2.8. **Co-indexing; rendering directly into $\mathsf{L}^{\lambda}_{\mathbf{ar}}$.** Roughly speaking, co-indexing occurs when the references of one or more indexical expressions in a term are identified with that of a subterm by the introduction of a bound variable which refers to all of them. Some examples of co-indexing in the $\lambda$-calculus:

$$\begin{aligned} &\text{John loves himself} \xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{himself}) \\ &\xrightarrow{\text{co-index}}_{\lambda} \Big(\lambda(j)\mathsf{loves}(j, j)\Big)(\mathsf{John}) \end{aligned} \tag{12}$$

$$\begin{aligned} &\text{John kissed his wife} \xrightarrow{\text{formalize}} \mathsf{kissed}(\mathsf{John}, \mathsf{wife}(\mathsf{his})) \\ &\xrightarrow{\text{co-index}}_{\lambda} \Big(\lambda(j)\mathsf{kissed}(j, \mathsf{wife}(j))\Big)(\mathsf{John}) \end{aligned} \tag{13}$$

$$\begin{aligned} &\text{John loves his wife and he honors her} \\ &\xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{wife}(\mathsf{his})) \,\&\, \mathsf{honors}(\mathsf{he}, \mathsf{her}) \\ &\xrightarrow{\text{co-index}}_{\lambda} \lambda(j)\Big[\mathsf{loves}(j, \mathsf{wife}(j)) \,\&\, \mathsf{honors}(j, \mathsf{her})\Big](\mathsf{John}) \\ &\xrightarrow{\text{co-index}}_{\lambda} \lambda(j)\Big[\lambda(w)\Big(\mathsf{loves}(j, w) \,\&\, \mathsf{honors}(j, w)\Big)(\mathsf{wife}(j))\Big](\mathsf{John}). \end{aligned} \tag{14}$$

Co-indexing is part of the rendering operation, since whether and how it should be done is determined by the informal context discussed in Footnote 4. The analysis of the examples suggests that it may be viewed as a formal operation on terms, which

[18] The local reading of an intransitive verb $F : (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{t}}$ can be obtained from the modal version in the same way that we derived the de-re reading from the de-dicto reading of modal operators in §2.5:

$$F_1(x)(a) = F(\text{dere}(x, a))(a).$$

Which of the two should be used in each case is not a matter of logic but one of language, and so it must be done at the rendering stage; we choose the modal rendering for rising temperatures and the local one for running presidents because they give us the desired meanings (and truth values).

completes the rendering after an initial formalization that leaves the indexicals alone; symbolically

$$\xrightarrow{\text{render}} = \xrightarrow{\text{formalize}} + \xrightarrow{\text{co-index}}_1 + \cdots + \xrightarrow{\text{co-index}}_k .$$

I will not attempt to define this operation here, since it is not clear at this point how to do it in full generality. What is worth noticing, however, is that *the recursion construct provides an alternative way to co-index* which, in fact, *may lead to essentially new renderings* of simple English sentences. For the examples above:

$$\begin{aligned} &\text{John loves himself} \xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{himself}) \\ (15)\qquad &\xrightarrow{\text{co-index}}_{\text{ar}} \mathsf{loves}(j, j) \text{ where } \{j := \mathsf{John}\} \\ &\text{John kissed his wife} \xrightarrow{\text{formalize}} \mathsf{kissed}(\mathsf{John}, \mathsf{wife}(\mathsf{his})) \\ (16)\qquad &\xrightarrow{\text{co-index}}_{\text{ar}} \mathsf{kissed}(j, \mathsf{wife}(j)) \text{ where } \{j := \mathsf{John}\} \end{aligned}$$

John loves his wife and he honors her

$$\begin{aligned} &\xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{wife}(\mathsf{his})) \,\&\, \mathsf{honors}(\mathsf{he}, \mathsf{her}) \\ &\xrightarrow{\text{co-index}}_{\text{ar}} \mathsf{loves}(j, \mathsf{wife}(j)) \,\&\, \mathsf{honors}(j, \mathsf{her}) \text{ where } \{j := \mathsf{John}\} \\ (17)\qquad &\xrightarrow{\text{co-index}}_{\text{ar}} \Big(\mathsf{loves}(j, w) \,\&\, \mathsf{honors}(j, w) \text{ where } \{w := \mathsf{wife}(j)\}\Big) \\ &\qquad\qquad \text{where } \{j := \mathsf{John}\} \\ (18)\qquad &\approx \mathsf{loves}(j, w) \,\&\, \mathsf{honors}(j, w) \text{ where } \{w := \mathsf{wife}(j), j := \mathsf{John}\} \end{aligned}$$

It will turn out that, naturally enough, (12) and (15) are referentially synonymous, but (13) is not referentially synonymous with (16), and neither is (14) referentially synonymous with (17) or its synonym (18), which we have included for clarity. Moreover, we will show in §3.25 that *the terms in* (16)*, and* (17) *are not referentially synonymous with any explicit terms*, i.e., their referential intensions can only be expressed using the recursion construct. It is a matter for investigation, of course, whether these $\mathsf{L}^{\lambda}_{\text{ar}}$ terms express "more naturally" (or, more to the point, more usefully for further processing) the English sentences that they render; we will return to this point in §3.26.

§2.9. **Proper nouns, demonstratives and quantifiers**. One of the most original innovations in Montague [1973] is the interpretation of "John", "I" and "the blond" by quantifiers, of type $\tilde{\mathsf{q}} \equiv (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{t}}$ (in the present system), so that he gets the uniform renderings,

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

Here Montague interprets "John" by the evaluation function,

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

In addition to the obvious advantage of assigning similar formal renderings to similar constructions of natural language, the device also facilitates greatly the operation of *coordination*, which we will discuss in §2.10. I will not adopt it, however, because the Montague renderings *produce the wrong logical form* for the

syntactical expressions that they purport to formalize, and thus *lose the intended meaning*. Specifically, we will show in §3.27 that the Montague renderings

$$\text{The evening star is the morning star} \xrightarrow{\text{render}} \mathsf{ES}_{\text{Mont}}(\lambda(u)\mathsf{MS}_{\text{Mont}}(\lambda(v)(u = v))),$$
$$\text{The morning star is the evening star} \xrightarrow{\text{render}} \mathsf{MS}_{\text{Mont}}(\lambda(u)\mathsf{ES}_{\text{Mont}}(\lambda(v)(u = v)))$$

of the classic Frege example are not referentially synonymous, as, of course, they should be.[19] The more natural renderings

$$\mathsf{ES} = \mathsf{MS},\ \mathsf{MS} = \mathsf{ES}$$

which come from the typing we have adopted are referentially synonymous.

It is not hard to formulate rules for rendering which avoid unnecessary type-raising and give plausible results for (at least) simple expressions which involve singular terms or quantifiers (or both). The basic technique is known as *type-driven rendering* (or translation), cf. Klein and Sag [1985] or the more recent textbook Heim and Kratzer [1998][Chapter 3], where it is applied using phrase structure trees to represent meanings. For example, for English phrases of the form

$$A\ \phi \text{ with } A \xrightarrow{\text{render}} \overline{A},\ \phi \xrightarrow{\text{render}} \overline{\phi} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}},$$

like "John runs" or "every man runs",

$$\text{if } \overline{A} : \tilde{e},\ \text{set } A\ \phi \xrightarrow{\text{render}} \overline{\phi}(\overline{A}),\ \text{and if } \overline{A} : \tilde{q},\ \text{set } A\ \phi \xrightarrow{\text{render}} \overline{A}(\overline{\phi}),$$

which in the examples gives the correct readings

$$\text{John runs} \xrightarrow{\text{render}} \mathsf{runs}(\mathsf{John}) \quad \text{and} \quad \text{every man runs} \xrightarrow{\text{render}} \mathsf{every}(\mathsf{man})(\mathsf{runs}).$$

A similar, somewhat more complex rendering rule (with four cases) can be given for English expressions

$$A\ \phi\ B$$

where $\phi$ is a transitive verb, like "John loves every woman" or "every man loves John's wife", although the matter is certainly not that simple for more complex syntactical expressions.

For our purposes here, the main lesson is that *meaning* (intuitively understood) *must be seriously considered in the rendering process*—simply "getting the right denotation" is not enough; and that the subsequent, formal computation of referential intensions and synonymies provides some clues as to whether the informal meaning was captured by the proposed rendering.

§2.10. **Coordination**. "John and Mary entered the room" does not have quite the same meaning as "John entered the room and Mary entered the room", because (for one thing) they do not have the same logical form: the first is a predication, while the second is a conjunction.[20] Similarly, "The temperature is 90° and rising" (a predication) is not synonymous with "The temperature is 90° and it is rising",

[19] I am assuming here (and in the sequel) that "The evening star is the morning star" is an *identity statement*, as Frege understood it, and so intuitively synonymous with its converse. Those who read Frege differently or do not agree with him on this, may want to use the example

one plus five equals two times three

whose status as an identity statement is hard to deny and with which the same point can be made.

[20] Cf. Ouhalla [1994][Section 2.8].

which is a conjunction. To capture these distinctions we must *coordinate* "John" and "Mary", put them together into a single object—which, however, cannot now be a singular object of type $\tilde{\mathsf{e}}$, but must be a quantifier; and (what seems easier), we must combine "is 90°" and "rising" into a single relation. The abstraction construct is a powerful tool for defining these coordination operations in combination with type-driven rendering, and it is well understood how they can be expressed in the $\lambda$-calculus. In the spirit of the preceding two paragraphs, however, it may be worth listing here some alternative renderings directly into $\mathsf{L}^{\lambda}_{\mathsf{ar}}$, which use the recursion construct and (in some cases) produce substantially simpler meanings.[21]

**(A)** If $X_i \xrightarrow{\text{render}} \overline{X}_i : \tilde{\mathsf{e}}$, set

$$X_1 \text{ and } X_2 \xrightarrow{\text{render}} \lambda(r)(r(x_1)\,\&\,r(x_2)) \text{ where } \{x_1 := \overline{X}_1, x_2 := \overline{X}_2\}.$$

Thus

$$\text{John and Mary} \xrightarrow{\text{render}} \lambda(r)(r(x_1)\,\&\,r(x_2)) \text{ where } \{x_1 := \mathsf{John}, x_2 := \mathsf{Mary}\} : \tilde{\mathsf{q}}.$$

**(B)** If $X \xrightarrow{\text{render}} \overline{X} : \tilde{\mathsf{e}}$ and $Q \xrightarrow{\text{render}} \overline{Q} : \tilde{\mathsf{q}}$, set

$$X \text{ and } Q \xrightarrow{\text{render}} \lambda(r)(r(x)\,\&\,q(r)) \text{ where } \{x := \overline{X}, q := \overline{Q}\} : \tilde{\mathsf{q}},$$

so that

the teacher and every student

$$\xrightarrow{\text{render}} \lambda(r)(r(x)\,\&\,q(r)) \text{ where } \{x := \mathsf{the}(\mathsf{teacher}), q := \mathsf{every}(\mathsf{student})\}.$$

**(C)** If $Q_i \xrightarrow{\text{render}} \overline{Q}_i : \tilde{\mathsf{q}}$, set

$$Q_1 \text{ and } Q_2 \xrightarrow{\text{render}} \lambda(r)(q_1(r)\,\&\,q_2(r)) \text{ where } \{q_1 := \overline{Q}_1, q_2 := \overline{Q}_2\} : \tilde{\mathsf{q}},$$

so that

some boy and every girl

$$\xrightarrow{\text{render}} \lambda(r)(b(r)\,\&\,g(r)) \text{ where } \{b := \mathsf{some}(\mathsf{boy}), g := \mathsf{every}(\mathsf{girl})\}.$$

**(D)** If $P_i \xrightarrow{\text{render}} \overline{P}_i : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$, set

$$P_1 \text{ and } P_2 \xrightarrow{\text{render}} \lambda(i)(p_1(i)\,\&\,p_2(i)) \text{ where } \{p_1 := \overline{P}_1, p_2 := \overline{P}_2\} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}},$$

so that (adding an application to get the Partee example)

The temperature is 90° and rising

$$\xrightarrow{\text{render}} \Big(\lambda(t)(n(t)\,\&\,r(t)) \text{ where } \{n := \lambda(x)[x = 90^\circ], r = \mathsf{rises}\}\Big)(\mathsf{the}(\mathsf{temperature})).$$

[21] Like co-indexing, coordination should be defined as a formal operation on terms to be performed after an initial formalization; whether it should come before or after co-indexing (or whether that matters) is a matter for investigation. The examples here do not cover the most general case, which is quite complex; see also §3.18.

**§3. Referential intensions and referential synonymy.** The technical work in this, the main section of the paper, will proceed in three parts, as follows.

**I. Reduction, irreducibility, canonical forms** (§3.5–§3.15 ). We will define a binary relation $\Rightarrow$ of *reduction* between terms, so that, intuitively,

$$A \Rightarrow B \iff A \equiv_c B \quad (A \text{ is congruent with } B)$$
$$\text{or } A \text{ and } B \text{ have the same meaning}$$
$$\text{and } B \text{ expresses that meaning "more simply".}$$

The disjunction is needed because some terms will not be assigned meanings, but the reduction calculus will still apply to them. We set

$$A \text{ is irreducible} \iff \text{for all } B, \text{ if } A \Rightarrow B, \text{ then } A \equiv_c B. \tag{19}$$

Irreducible terms which have meaning, express their meaning "as simply as possible".

We will then outline a proof of the following, simple but basic result of the paper:

§3.1. **Canonical Form Theorem**. *For each term $A$, there is an irreducible term*

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

*such that $A \Rightarrow \mathrm{cf}(A)$; moreover, $\mathrm{cf}(A)$ 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 call $\mathrm{cf}(A)$ the *canonical form of* $A$ and we write

$$A \Rightarrow_{\mathrm{cf}} B \iff \mathrm{cf}(A) \equiv_c B.$$

The terms $A_0, A_1, \ldots, A_n$ are the *parts* of $A$, and $A_0$ is its *head*. It will be convenient to employ the notational convention

$$A \text{ where } \{\ \} \equiv A$$

introduced in (4), which allows us to assume that all canonical forms look like recursive terms—perhaps with an empty body:

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

The definition of reduction is by ten, simple *reduction rules*, and the computation of canonical forms is effective.

§3.2. **Logical form and syntactic synonymy**. The canonical form of a term $A$ expresses the meaning of $A$ directly in terms of the primitives of the language and the vocabulary, and it gives a plausible explication of the *logical form* of the natural language phrase rendered by $A$—if $A$ renders a phrase. Two terms $A$ and $B$ are *syntactically synonymous* if their canonical forms are congruent, in symbols

$$A \approx_s B \iff \mathrm{cf}(A) \equiv_c \mathrm{cf}(B). \tag{20}$$

This stands for synonymy on the basis of logical form alone. We will establish several natural properties of syntactic synonymy.

**II. Referential intensions** (§3.16–§3.20). Variables and some very simple, *immediate terms* have no meaning, they denote *immediately*. Constants, on the other hand, denote *directly*, but they have meanings (albeit trivial ones), and they contribute differently to the meanings of the terms in which they occur. The distinction between immediate and direct reference is a central feature of this theory and we will discuss it in §3.7, where immediate terms are defined, and in Section 4, especially §4.8.

If $A$ is *proper* (i.e., not immediate) and

$$\mathsf{cf}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \dots, p_n := A_n\} \qquad (n \geq 0),$$

then the *referential intension* $\mathrm{int}(A)$ of $A$ is (intuitively) the abstract algorithm which computes for each assignment g the denotation $\mathrm{den}(A)(\mathrm{g})$, as that was described in Case (D4) of §1.4 and the example following it. The precise definition is given in §3.16.

The parts $A_0, A_1, \dots, A_n$ of a term $A$ are *explicit, irreducible terms*, whose meanings (if they exist) are exhausted by their denotations; and the assignments of denotations to these terms may be regarded as the *relevant, basic facts* needed for the determination of the denotation of $A$. The referential intension $\mathrm{int}(A)$ "codifies" in a mathematical object these facts and the natural process by which $\mathrm{den}(A)(\mathrm{g})$ is computed from them.

§3.3. **Canonical forms and truth conditions**. If $A$ is a Carnap intension, then its canonical form may also be viewed as a generalized (or just precise) version of Davidson's *set of truth conditions* for $A$, whose relation to meaning is described as follows in Davidson [1967]:

> ... the obvious connection between a definition of truth of the kind Tarski has shown how to construct, and the concept of meaning ... is this: the definition works by giving necessary and sufficient conditions for the truth of every sentence, and to give truth conditions is a way of giving the meaning of a sentence.

Davidson does not take the next step, which is to extract a semantic object from these truth conditions and call it "the meaning of $A$", and, in fact, he denies that this step is useful or even possible. Despite this important difference, it is quite clear that the approach to language in this paper is very close to Davidson's, and can even be viewed as incorporating Davidson's basic insights into a "Fregean theory of meaning" (with meanings!) whose possibility Davidson doubts.

**III. Referential synonymy** (§3.4–§3.27). Two proper terms are *referentially synonymous* if their referential intensions are *naturally isomorphic*, so that they model—*they are*, from the mathematical point of view—identical algorithms. It is also convenient to call two immediate terms $X$ and $Y$ referentially synonymous if they have the same denotation for all assignments to the variables. We have already introduced in (7) the notation

$$A \approx B \iff A \text{ and } B \text{ are referentially synonymous.}$$

The precise, general definition of natural isomorphism in §3.20 is a bit technical, but it implies a very simple characterization of the referential synonymy relation on terms:

(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

$$A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\} \Rightarrow B_0 \text{ where } \{p_1 := B_1, \ldots, p_n := B_n\}$$

TABLE 4. The reduction calculus: congruence, transitivity, compositionality.

§3.4. **Referential Synonymy Theorem**. *Two terms $A, B$ are referentially synonymous if and only if*

$$A \Rightarrow_{\text{cf}} A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\},$$
$$B \Rightarrow_{\text{cf}} B_0 \text{ where } \{p_1 := B_1, \ldots, p_n := B_n\},$$

*for some $n \geq 0$ and suitable, $A_0, A_1, \ldots, A_n, B_0, B_1, \ldots, B_n$, so that*

$$\models A_i = B_i \quad (i = 0, 1, \ldots, n).$$

*In particular,*

$$A \Rightarrow B \Longrightarrow A \approx_s B \Longrightarrow A \approx B.$$

The result reduces referential synonymy to a system of (effectively determined) denotational identities between explicit, irreducible terms, and it is at the heart of the proposed theory of meaning. Notice that $n = 0$ is allowed in this theorem (by the convention (4)) and occurs when $A$ and $B$ are explicit, irreducible terms: such terms are synonymous exactly when they have the same denotation.

Referential synonymy differs from syntactic synonymy in that it takes into account the intended interpretation of the constants and the constructs of $\mathsf{L}^{\lambda}_{\text{ar}}(K)$; if, for example, $\text{den}(\mathsf{a}) = \text{den}(\mathsf{b})$ for two distinct constants, then $a \approx b$ but $\mathsf{a} \not\approx_s \mathsf{b}$, and if wife is a constant, then $\text{wife} \approx \lambda(x)\text{wife}(x)$ but $\text{wife} \not\approx_s \lambda(x)\text{wife}(x)$.[22]

We now turn to the technical development of parts I – III.

§3.5. **The reduction calculus: congruence, transitivity, compositionality**. The first five rules of the Reduction Calculus are listed in Table 4, and they simply insure that the reduction relation is transitive and compositional, and that it extends the congruence relation. They do not produce by themselves any non-trivial reductions.

§3.6. **The reduction rules for recursion.** These are listed in Table 5, and they allow us to combine recursive definitions. In stating them I have used the abbreviations

$$\vec{p} := \vec{A} \text{ for } p_1 := A_1, \ldots, p_n = A_n,$$
$$\vec{q} := \vec{B} \text{ for } q_1 := B_1, \ldots, q_m = B_m,$$

[22]There is also an intermediate notion $\approx_\ell$ of *logical synonymy* which takes into account the meaning of the constructs of $\mathsf{L}^{\lambda}_{\text{ar}}(K)$ but not that of the constants, so that (with the same assumptions) $\mathsf{a} \not\approx_\ell \mathsf{b}$ but $\text{wife} \approx_\ell \lambda(x)\text{wife}(x)$.

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

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

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

TABLE 5. The reduction calculus: rules for recursion.

where it is assumed that $p_1, \ldots, p_n, q_1, \ldots q_m$ are distinct locations, and I have omitted some (mostly obvious) restrictions on occurrences of variables. Here they are in full, with some examples.

**The head-rule** (head). *Restriction*: *No $p_i$ occurs free in any $B_j$*.[23] Thus:

$$\Big(\mathsf{loves}(j, w) \text{ where } \{w := \mathsf{wife}(p), p := \mathsf{Paul}\}\Big) \text{ where } \{j := \mathsf{John}\}$$
$$\Rightarrow \mathsf{loves}(j, w) \text{ where } \{w := \mathsf{wife}(p), p := \mathsf{Paul}, j := \mathsf{John}\}.$$

**The Bekič-Scott rule** (B-S). *Restriction*: *No $q_j$ occurs free in any $A_i$*. Example:

$$\mathsf{loves}(j, w) \text{ where } \{w := \Big(\mathsf{wife}(p) \text{ where } \{p := \mathsf{Paul}\}\Big), j := \mathsf{John}\}$$
$$\Rightarrow \mathsf{loves}(j, w) \text{ where } \{w := \mathsf{wife}(p), p := \mathsf{Paul}, j := \mathsf{John}\}.$$

The examples have been silly because the rules we have introduced so far don't do much reducing: basically they say that nested occurrences of "where" can be "flattened out", which is an obvious move. The next rule is not so innocuous.

**The recursion-application rule** (recap). *Restriction*: *No $p_i$ occurs free in $B$*. For an example, let[24]

$$A \equiv (h = s) \text{ where } \{h := \mathsf{He}, s := \mathsf{Scott}\},$$

a term which is synonymous with

$$\text{He is Scott},$$

as we will see after the next group of reductions. The term $A$ is a closed Carnap intension which is true in a state $a$ exactly when

$$\mathrm{He}(a) = \mathrm{Scott}(a).$$

By the recap rule, for any state variable $x$,

$$A(x) \Rightarrow B \equiv (h = s)(x) \text{ where } \{h := \mathsf{He}, s := \mathsf{Scott}\},$$

[23]The restriction implies, in particular, that the recursive term on the right is acyclic, if the given terms are. This must be formally checked for each of the rules, but it is quite simple in all cases and I will not bring it up again.

[24]As everybody knows, "Scott" is a rigid constant which denotes Sir Walter Scott in every state.

and it is clear that $\mathrm{den}(A(x))(\mathrm{g}) = \mathrm{den}(B)(\mathrm{g})$ for every assignment g, but: notice that $x$ occurs only in the head $(h = s)(x)$ of $B$. To compute the denotation of $A(x)$ by the recipe suggested by the form of $B$, we follow the following

*Procedure* 1.

Stage 1: Set $h :=$ He, $s :=$ Scott.

Stage 2: Set $a := \mathrm{g}(x)$; if $h(a) = s(a) =$ Sir Walter, give the value 1, otherwise give the value 0.

Some people might compute $\mathrm{den}(A(x))(\mathrm{g})$ by the following, somewhat different procedure

*Procedure* 1′.

Stage 1′. Set $a := \mathrm{g}(x)$.

Stage 2′: $h' :=$ He$(a)$, $s' :=$ Scott$(a) =$ Sir Walter.

Stage 3′: If $h' = s' =$ Sir Walter, give the value 1, otherwise give the value 0.

Not much difference between the two, perhaps, but those who use Procedure 1′ never encounter the "full" functions "He" and " Scott", only their values in the particular state $a$. Put another way, in terms of meanings: the (full) meaning of "He" and that of "Scott" are parts of the meaning of $B$ (and hence of $A(x)$) for the notion of meaning that will be determined by this reduction relation.

Only two more rules remain, but they are the ones which do most of the work. The first of these depends for its formulation on the notion of *immediacy* and—for the first time—differentiates between pure and recursion variables!

§3.7. **Immediate terms**. Variables (of either kind) are immediate, and so are "generalized variables" of the following forms

$$X :\equiv v_i \mid p \mid p(v_1, \ldots, v_n) \mid \lambda(u_1, \ldots, u_m)p \mid \lambda(u_1, \ldots, u_m)(p(v_1, \ldots, v_n))$$
$$(p \text{ a location}, v_i, u_j \text{ pure})$$

The key point is that if $p$ is a location of function type and $u, v$ are pure variables, then $p(v)$ is immediate while $u(v)$ is not. Terms which are not immediate are *proper*.

In computational terms, we can think of a location $p : (\sigma \to \tau)$ of function type as having its entire graph (table, course of values) stored "in the machine", as soon as it is specified by an assignment, and then any value $p(v)$ of it is simply *read*, as is any value of $\lambda(u, v)p(u, z, w)$; a pure variable $u : (\sigma \to \tau)$ is represented by a *port*, and to access a value $u(v)$ we must make a *call* to that port, which then provides the required value $u(v)$.

Constants are proper, in any type; to understand this don't think of "Scott" (whose "computation" appears to be trivial), think of "$\pi$" which calls for the non-trivial recomputation of the number $\pi$ each time it is encountered. It is standard advice to beginning programmers to set up an assignment $p := \pi$ and then replace "$\pi$" by "$p$" throughout their program, if "$\pi$" occurs many times. The new program expresses a more efficient algorithm, which computes $\pi$ only once, stores the value, and then just reads it each time it is needed.

§3.8. **The application rule (ap)**. This is stated in Table 6. To understand why the non-immediacy restriction is needed, consider the example

$$\text{John runs} \xrightarrow{\text{render}} \mathsf{runs}(\mathsf{John}) \Rightarrow \mathsf{runs}(j) \text{ where } \{j := \mathsf{John}\}.$$

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

TABLE 6. The reduction calculus: the application rule.

If the unrestricted rule preserved meanings, we would have

$$\mathsf{runs}(j) \Rightarrow \mathsf{runs}(j')\ \mathsf{where}\ \{j' := j\} \quad \text{(Caution: this is false!)}$$

and then we could continue with the reductions

$$\begin{aligned}\mathsf{runs}(\mathsf{John}) &\Rightarrow \mathsf{runs}(j)\ \mathsf{where}\ \{j := \mathsf{John}\}\\ &\Rightarrow \Big(\mathsf{runs}(j')\ \mathsf{where}\ \{j' := j\}\Big)\ \mathsf{where}\ \{j := \mathsf{John}\} && \text{(rep3)}\\ &\Rightarrow \mathsf{runs}(j')\ \mathsf{where}\ \{j' := j, j := \mathsf{John}\}, && \text{(head)}\end{aligned}$$

so that, in particular,

$$\mathsf{runs}(j)\ \mathsf{where}\ \{j := \mathsf{John}\} \approx \mathsf{runs}(j')\ \mathsf{where}\ \{j' := j, j := \mathsf{John}\}.$$

But this is surely not right, at least if we allow for some computational aspect in the notion of meaning: because it takes three steps to compute the right-hand-side (as we have been doing these computations), while two suffice for the left. Moreover, if we did this several times, we would get arbitrarily long terms of the form

$$\mathsf{runs}(j_1)\ \mathsf{where}\ \{j_1 := j_2, j_2 := j_3, \ldots, j_n := j_{n+1}, j_{n+1} := \mathsf{John}\},$$

all of them allegedly synonymous with "John runs", which does not look right.

Finally, the application rule is consistent with our intuitions about synonymy only because of our disallowing the interpretation of the constants of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ by *propositional attitudes*, like knowledge or belief. If reduction implies synonymy and we had a constant $\mathsf{I\ know}$ in $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, then it cannot be that for any closed term $A : \mathsf{t}$,

$$\text{I know that } A \xrightarrow{\text{render}} \mathsf{I\ know}(A) \Rightarrow \mathsf{I\ know}(p)\ \mathsf{where}\ \{p := A\};$$

because $p : \mathsf{t}$ in this term, which means that the constant $\mathsf{I\ know}$ denotes a function $K : \mathbb{T}_{\mathsf{t}} \to \mathbb{T}_{\mathsf{t}}$ such that $K(1) = 1$, since "I know that $1 + 1 = 2$"; and hence "I know that $A$" for every true proposition $A$, which is absurd.[25]

§3.9. **The canonical form of "John loves Mary"**. The last (still missing) reduction rule does not affect $\lambda$-free terms, and so $\mathsf{runs}(j)\ \mathsf{where}\ \{j := \mathsf{John}\}$ is irreducible, since (by a simple inspection) none of the nine rules we have listed so far other than (cong) can be applied to it. This means that the single application

$$\mathsf{runs}(\mathsf{John}) \Rightarrow_{\mathrm{cf}} \mathsf{runs}(j)\ \mathsf{where}\ \{j := \mathsf{John}\}$$

of the (ap) rule gives us the canonical form of $\mathsf{tall}(\mathsf{John})$. Let's write out one more complete reduction to canonical form which is just as trivial but illustrates the use of the recursion rules:

$$\text{John loves Mary} \xrightarrow{\text{render}} \mathsf{loves}(\mathsf{John}, \mathsf{Mary}) \equiv \mathsf{loves}(\mathsf{John})(\mathsf{Mary})$$

[25]The theory of referential intensions has some implications for the meaning of propositional attitudes, but they are not in this paper, except for the brief remark in §4.9.

($\lambda$-rule) $\lambda(u)\Big(A_0 \text{ where } \{p_1 := A_1, \dots, p_n := A_n\}\Big)$
$\Rightarrow \lambda(u)A'_0 \text{ where } \{p'_1 := \lambda(u)A'_1, \dots, p'_n := \lambda(u)A'_n\}$

where for $i = 1, \dots, m$, $p'_i$ is a fresh location and $A'_i$ is defined by the replacement

$$A'_i :\equiv A_i\{p_1 :\equiv p'_1(u), \dots, p_n :\equiv p'_n(u)\}.$$

TABLE 7. The $\lambda$-rule.

$$\begin{aligned}
\mathsf{loves}(\mathsf{John}) &\Rightarrow_{\mathrm{cf}} \mathsf{loves}(j) \text{ where } \{j := \mathsf{John}\} && \text{(ap)}\\
\mathsf{loves}(\mathsf{John})(\mathsf{Mary}) &\Rightarrow \Big(\mathsf{loves}(j) \text{ where } \{j := \mathsf{John}\}\Big)(\mathsf{Mary}) && \text{(rep1)}\\
&\Rightarrow \mathsf{loves}(j)(\mathsf{Mary}) \text{ where } \{j := \mathsf{John}\} && \text{(recap)}\\
&\Rightarrow \Big(\mathsf{loves}(j)(m) \text{ where } \{m := \mathsf{Mary}\}\Big)\\
&\qquad\qquad \text{where } \{j := \mathsf{John}\} && \text{(ap,rep3)}\\
&\Rightarrow_{\mathrm{cf}} \mathsf{loves}(j, m)\\
&\qquad\qquad \text{where } \{j := \mathsf{John}, m := \mathsf{Mary}\} && \text{(head,cong)}
\end{aligned}$$

In the same way, we can compute

(21) $\qquad \text{He is Scott} \Rightarrow_{\mathrm{cf}} (h = s) \text{ where } \{h := \mathsf{He}, s := \mathsf{Scott}\},$

and assuming (for simplicity) that the language has constants for addition, multiplication and for the first few numbers,

(22) $\quad 1 + 5 = 2 \times 3$
$\Rightarrow_{\mathrm{cf}} (a = b) \text{ where } \{a := o + f, o := 1, f := 5, b := t \times r, t := 2, r := 3\}.$

§3.10. **The $\lambda$-rule**. To motivate the $\lambda$-rule in Table 7, consider the Carnap intension

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

The crucial part is the $\lambda$-term to which the quantifier (every)(man) is applied, and for that we first reduce the matrix:

$$B \equiv \mathsf{danced}(u, \mathsf{wife}(u)) \Rightarrow_{\mathrm{cf}} \mathsf{danced}(u, w) \text{ where } \{w := \mathsf{wife}(u)\}.$$

The standard computation of the value of $B$ requires us to set

$$w := \mathrm{wife}(u)$$

for any $u$, so what is really being computed is the function $w'(u) = \mathrm{wife}(u)$—which is, in fact, what we need for the subsequent application of the quantifier; and the simplest way to effect this is to set

$$\lambda(u)(B) \Rightarrow \lambda(u)\mathsf{danced}(u, w'(u)) \text{ where } \{w' := \lambda(u)\mathsf{wife}(u)\}.$$

The reduction calculus does not justify the next step we would like to take,

$$\lambda(u)\mathsf{wife}(u) \Rightarrow \mathsf{wife},$$

but we will see later that

$$\lambda(u)\mathsf{wife}(u) \approx \mathsf{wife},$$

*if "*wife*" is a constant*; if, however, it is an abbreviation introduced by

$$\mathsf{wife}(u) \equiv \mathsf{the}(\lambda(v)\mathsf{married}(u,v)),$$

then we can use the $\lambda$-rule again to compute

$$\begin{aligned}\lambda(u)\mathsf{the}(\lambda(v)\mathsf{married}(u,v)) &\Rightarrow \lambda(u)[\mathsf{the}(w) \text{ where } \{w := \lambda(v)\mathsf{married}(u,v)\}]\\ &\Rightarrow \lambda(u)\mathsf{the}(w(u)) \text{ where } \{w := \lambda(u)\lambda(v)\mathsf{married}(u,v)\},\end{aligned}$$

and *if now* "married" *is a constant*, we have

$$\lambda(u)\lambda(v)\mathsf{married}(u,v) \approx \mathsf{married},$$

so that at the level of synonymy we get

$$\begin{aligned}\lambda(u)(B) &\Rightarrow \lambda(u)\mathsf{danced}(u,w'(u)) \text{ where } \{w' := \lambda(u)\mathsf{wife}(u)\}\\ &\approx \lambda(u)\mathsf{danced}(u,w'(u)) \text{ where } \{w' := \lambda(u)\mathsf{the}(w(u)), w := \mathsf{married}\}.\end{aligned}$$

This completes the definition of the reduction relation. We claim that it preserves meaning, so it had better preserve at least denotations:

§3.11. **Theorem**. *If $A \Rightarrow B$, then $\models A = B$.*

PROOF is simple, by induction on the definition of the reduction relation. ⊣

It is also easy to read off the reduction rules a simple characterization of irreducible terms, defined in (19):

§3.12. **Theorem**. (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 $\{p_1 := A_1, \ldots, p_n := A_n\}$ is irreducible if and only all the parts $A_0, \ldots, A_n$ are explicit and irreducible.*
The proof is simple, by inspection of the reduction rules.

§3.13. **Canonical forms**. We define the canonical form $\mathrm{cf}(A)$ of each term $A$ by the following recursion on terms, assuming in each of the clauses that all bound locations are distinct and distinct from all the free locations. (This can be insured by making suitable alphabetic changes on the bound variables of the given terms before we apply each clause, if needed.)

(CF1) $\mathrm{cf}(\mathsf{c}) :\equiv \mathsf{c}$ ($\equiv \mathsf{c}$ where $\{\ \}$); $\quad \mathrm{cf}(x) :\equiv x$ ($\equiv x$ where $\{\ \}$).

(CF2) Suppose $\mathrm{cf}(A) \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$ $(n \geq 0)$. If $X$ is immediate, then

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

and if $B$ is proper and $\mathrm{cf}(B) \equiv B_0$ where $\{q_1 := B_1, \ldots, q_n := B_m\}$, then

$$\mathrm{cf}(A(B)) :\equiv A_0(q_0) \text{ where } \{p_1 := A_1, \ldots, p_n := A_n, \\ q_0 := B_0, q_1 := B_1, \ldots, q_m := B_m\}.$$

(CF3) For any pure variable $u$, if

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

then,

$$\mathrm{cf}(\lambda(u)A) :\equiv \lambda(u)A'_0 \text{ where } \{p'_1 := \lambda(u)A'_1, \ldots, p'_n := \lambda(u)A'_n\},$$

where (as in the $\lambda$-rule for reduction) each $p'_i$ is a fresh location and

$$A'_i \equiv A_i\{p_1 :\equiv p'_1(u), \ldots, p_n :\equiv p'_n(u)\}.$$

(CF4) If $A \equiv A_0$ where $\{p_1 := A_1, \ldots, p_n := A_n\}$ with $n \geq 0$ and if, for $i = 0, \ldots, n$,

$$\mathrm{cf}(A_i) \equiv A_{i,0} \text{ where } \{p_{i,1} := A_{i,1}, \ldots, p_{i,k_i} := A_{i,k_i}\} \qquad (k_i \geq 0),$$

then

$$\mathrm{cf}(A) :\equiv A_{0,0} \text{ where } \{ \quad p_{0,1} := A_{0,1}, \ldots, p_{0,k_0} := A_{0,k_0}, \\ p_1 := A_{1,0}, p_{1,1} := A_{1,1}, \ldots, p_{1,k_1} := A_{1,k_1}, \\ \vdots \\ p_n := A_{n,0}, p_{n,1} := A_{n,1}, \ldots, p_{n,k_n} := A_{n,k_n}\}.$$

In the next result we summarize the basic properties of canonical forms which, in particular, provide a proof of the Canonical Form Theorem §3.1.

§3.14. **Theorem**. *For every term $A$*:
(1) *The canonical form of $A$ is a term*

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

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

OUTLINE OF PROOF. (1) is verified easily, by inspection of the reduction rules, and (2) is also very simple, by induction on the term $A$. (3) is verified by an induction on the characterization of explicit, irreducible terms given in Theorem §3.12. It applies to immediate terms, which are explicit and irreducible. The crucial (4) is proved by induction on the definition of the reduction relation, and it involves (unfortunately) a great deal of computation. Finally, for (5), if $B$ is irreducible, then $B \equiv \mathrm{cf}(B)$, by (3); and so if $A \Rightarrow B$, then $\mathrm{cf}(A) \equiv_c \mathrm{cf}(B) \equiv B$ by (4). ⊣

Next we summarize some of the basic properties of syntactic synonymy, defined in (20).

§3.15. **Theorem**. (1) *If $A \Rightarrow D$ and $B \Rightarrow D$ for some $D$, then $A \approx_s B$; and, similarly, if $D \Rightarrow A$ and $D \Rightarrow B$ for some $D$, then $A \approx_s B$.*

(2) *If $A \approx_s X$ for some immediate term $X$, then $A$ is also immediate and $A \equiv_c X$.*

(3) *$\approx_s$ is an equivalence relation on terms which extends congruence and respects application, $\lambda$-abstraction and the formation of recursive terms, i.e.,*

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

$$\frac{A_0 \approx_s B_0, \qquad A_1 \approx_s B_1, \qquad \dots, \qquad A_n \approx_s B_n}{A_0 \text{ where } \{p_1 := A_1, \dots p_n := A_n\} \approx_s B_0 \text{ where } \{p_1 := B_1, \dots, p_n := B_n\}}$$

(4) *If $z : \sigma$ is a constant* c*, or a variable of either kind, $C : \sigma$ is a proper term of the same type and the substitution $\{z :\equiv C\}$ is free in $A$, then*

$$A\{z :\equiv C\} \approx_s \Big(\mathrm{cf}(A)\Big)\{z :\equiv C\}.$$

These facts are also established by somewhat messy, long computations.

§3.16. **Referential intensions**. Let G be the set of all assignments to the variables, suppose that $A$ is a proper (non-immediate) term with canonical form

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

and for $i = 0, \dots, n$ set

$$\alpha_i(g, d_1, \dots, d_n) = \mathrm{den}(A_i)(\mathrm{g}\{p_1 := d_1, \dots, p_n := d_n\}).$$

The *referential intension* of $A$ is the tuple of functions

$$\mathrm{int}(A) = (\alpha_0, \alpha_1, \dots, \alpha_n).$$

For example, by §3.9, $\mathrm{int}(\mathsf{loves}(\mathsf{John}, \mathsf{Mary})) = (\alpha_0, \alpha_1, \alpha_2)$, where

$$\alpha_0(\mathrm{g}, j, m) = \mathrm{loves}(j, m),$$
$$\alpha_1(\mathrm{g}, j, m) = \mathrm{John},$$
$$\alpha_2(\mathrm{g}, j, m) = \mathrm{Mary}.$$

Notice that if $A, A_0 : \sigma$ and $A_i : \sigma_i$, then

$$\alpha_0 \ : \ \mathrm{G} \times \mathbb{T}_{\sigma_1} \times \cdots \times \mathbb{T}_{\sigma_n} \to \mathbb{T}_\sigma, \tag{23}$$

$$\text{and for } i = 1, \dots, n, \ \alpha_i \ : \ \mathrm{G} \times \mathbb{T}_{\sigma_1} \times \cdots \times \mathbb{T}_{\sigma_n} \to \mathbb{T}_{\sigma_i}. \tag{24}$$

Moreover, because of the acyclicity of $\mathrm{cf}(A)$, each $\alpha_i$ satisfies the following condition, for all $\mathrm{g}, d_1, \dots, d_n, d'_1, \dots, d'_n$, with $\mathrm{rank}(j) = \mathrm{rank}(p_j)$:

$$\text{if } d_j = d'_j \text{ for all } j \text{ such that } \mathrm{rank}(j) < \mathrm{rank}(i), \text{ then } \alpha_i(g, d_1, \dots, d_n) = \alpha_i(g, d'_1, \dots, d'_n). \tag{25}$$

§3.17. **Acyclic recursors**. A tuple of functions $(\alpha_0, \alpha_1, \dots, \alpha_n)$ which satisfies conditions (23)–(25) with some

$$\mathrm{rank} : \{1, \dots, n\} \to \{1, \dots, n\}$$

is called an *acyclic recursor* on G to $\mathbb{T}_\sigma$, with *internal type* $\sigma_1 \times \cdots \times \sigma_n$ and *dimension* $n \geq 0$. We write

$$\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_n) : \mathrm{G} \rightsquigarrow \mathbb{T}_\sigma \tag{26}$$

to indicate its *domain* of definition and *range* of values. The names are justified, because $\alpha$ determines (or *computes*) a function

$$\overline{\alpha} : \mathrm{G} \to \mathbb{T}_\sigma$$

as follows:

$$\overline{\alpha}(\mathrm{g}) = \alpha_0(\mathrm{g}, \overline{d}_1, \ldots, \overline{d}_n),$$

where, for each g, $\overline{d}_1, \ldots, \overline{d}_n$ are the unique solutions of the system of equations

$$d_i = \alpha_i(\mathrm{g}, d_1, \ldots, d_n) \quad (i = 1, \ldots, n),$$

guaranteed by the acyclicity condition. A recursor $\alpha = (\alpha_0)$ of dimension 0 is called *trivial*, as it is completely determined by the function $\overline{\alpha} = \alpha_0 : \mathrm{G} \to \mathbb{T}_\sigma$.

Thus the referential intension of a non-immediate term $A : \sigma$ is an acyclic recursor

$$\mathrm{int}(A) : \mathrm{G} \rightsquigarrow \mathbb{T}_\sigma,$$

and by clause (D4) of the definition of denotations §1.4 and Theorem §3.11, for every assignment g,

$$\overline{\mathrm{int}(A)}(\mathrm{g}) = \mathrm{den}(A)(\mathrm{g}),$$

i.e., the referential intension of $A$ computes its denotation.

§3.18. **Circuit diagrams**. The canonical form of a proper term $A$ and the recursor that it determines can be visualized as a *labeled directed graph*, a *circuit* really, with nodes the recursion variables $p_1, \ldots, p_n$ of $\mathrm{cf}(A)$; the part $A_i$ labeling the node $p_i$; and an arrow put from $p_i$ to $p_j$ if $p_j$ occurs in $A_i$. Figure 1 pictures this circuit for

$$A \equiv \text{John loves Mary and she loves him and every boy},$$

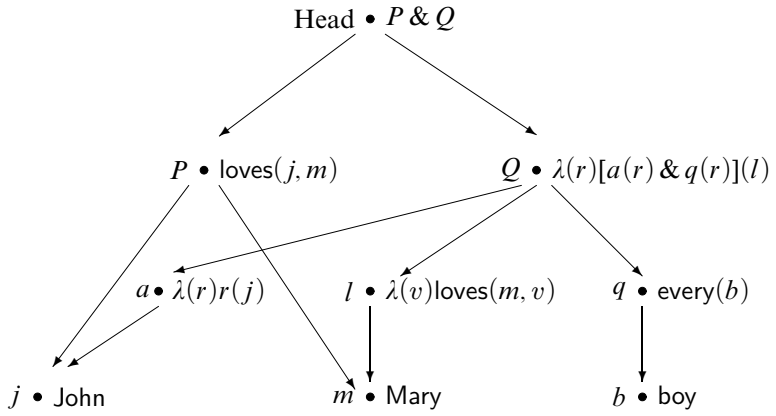
a reasonably complex sentence whose rendering involves both co-indexing and coordination.[26]

§3.19. **Algorithms and meanings as recursors**. The introduction promised to model the meaning of a term $A$ by an "abstract, idealized algorithm", but what has been delivered is an "acyclic recursor" $\mathrm{int}(A) = (\alpha_0, \alpha_1, \ldots, \alpha_n)$, a tuple of functions. It is not evident why—and in what sense—algorithms can be "faithfully represented" by recursors. This is discussed in Moschovakis [1998], and we will review some of these arguments in the last Section 5, where we will also examine why—and in what sense—meanings can be faithfully represented by algorithms, and exactly what kind of algorithms. Here we record only that we have constructed a precise model of the basic picture of a Fregean theory of meaning,

$$A \mapsto \mathrm{int}(A) \mapsto \mathrm{den}(A),$$

[26]The construction of this normal form assumes a coordination operation on formal terms which is executed after all co-indexing operations and which distinguishes immediate from proper arguments, so that

$$j \,\&\, \mathsf{every}(\mathsf{boy}) \xrightarrow{\text{coord}} \lambda(r)(r(j) \,\&\, q(r)) \text{ where } \{q := \mathsf{every}(\mathsf{boy})\}.$$




$A \Rightarrow_{\mathrm{cf}} P \,\&\, Q$ where $\{P := \mathsf{loves}(j,m), Q := \lambda(r)[a(r)\,\&\,q(r)](l), a := \lambda(r)r(j)$
$l := \lambda(v)\mathsf{loves}(m,v), q := \mathsf{every}(b), b := \mathsf{boy}, j := \mathsf{John}, m := \mathsf{Mary}\}$

FIGURE 1. *John loves Mary and she loves him and every boy*.

with $\mathrm{int}(A)$ an object which purports to model the meaning of $A$.

§3.20. **Natural recursor isomorphism**. An acyclic recursor (26) determines for each assignment g the system of mutual recursive equations

$$\left\{\begin{array}{l} d_1 \;=\; \alpha_1(\mathrm{g}, d_1, \ldots, d_n) \\ \qquad \vdots \\ d_n \;=\; \alpha_n(\mathrm{g}, d_1, \ldots, d_n) \end{array}\right. \tag{27}$$

whose unique solutions (along with the head $\alpha_0$) determine the value $\overline{\alpha}(\mathrm{g})$ as above. The order in which the equations are listed in (27) is of no consequence in this process of evaluation, and so it is natural to "identify" two recursors if they only differ in this respect. The precise definition is a bit technical:

Two acyclic recursors

$$\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_n),\ \beta = (\beta_0, \beta_1, \ldots, \beta_m) : \mathrm{G} \rightsquigarrow \mathbb{T}_\sigma$$

of respective internal types $\sigma_1 \times \cdots \times \sigma_n$ and $\tau_1 \times \cdots \times \tau_m$ and into the same output set $\mathbb{T}_\sigma$ are *naturally isomorphic*,[27] if they have the same dimension ($m = n$), and

[27]Natural isomorphism is the strictest equivalence relation among recursors which accords with our view of them as "the semantic content" of systems of recursive equations, with a head; the least-strict one is equality of denotations,

$$\alpha \sim \beta \iff (\forall \mathrm{g})[\overline{\alpha}(\mathrm{g}) = \overline{\beta}(\mathrm{g})].$$

In-between these two extremes, there are many interesting and useful ways to identify recursors, depending on what particular kind of "process" we are trying to model. I will confine myself here to natural isomorphism which captures the strictest notion of synonymy, but there are competing "identity conditions" for meanings which may prove useful upon further study.

$$\frac{A \approx_s B}{A \approx B}$$

$$A \approx A \qquad \frac{A \approx B}{B \approx A} \qquad \frac{A \approx B \qquad B \approx C}{A \approx C}$$

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

$$\frac{A_0 \approx B_0, \qquad A_1 \approx B_1, \qquad \dots, \qquad A_n \approx B_n}{A_0 \text{ where } \{p_1 := A_1, \dots p_n := A_n\} \approx B_0 \text{ where } \{p_1 := B_1, \dots, p_n := B_n\}}$$

$$\frac{\models C = D}{C \approx D}\ (*) \quad \text{hence:} \quad \frac{}{\Big(\lambda(u)C\Big)(v) \approx C\{u :\equiv v\}}\ (C \text{ e.i.})$$

e.i. : (congruent to) explicit, irreducible
$(*)$ : $C, D$ are both e.i., immediate or proper terms
$\models C = D \iff$ for all assignments g, $\mathrm{den}(C)(\mathrm{g}) = \mathrm{den}(D)(\mathrm{g})$
$u$ and $v$ are pure variables, and the substitution $C\{u :\equiv v\}$ is free

TABLE 8. The calculus of referential synonymy.

there is a permutation

$$\pi : \{0, 1, \dots, n\} \rightarrowtail \{0, 1, \dots, n\} \text{ with } \pi(0) = 0,$$

such that $\sigma_{\pi(i)} = \tau_i$ for $i = 1, \dots, n$ and

$$\alpha_{\pi(i)}(\mathrm{g}, d_1, \dots, d_n) = \beta_i(\mathrm{g}, d_{\pi(1)}, \dots, d_{\pi(n)}) \quad (\mathrm{g} \in \mathbf{G}, d_i \in \mathbb{T}_{\sigma_i}, i = 0, \dots, n).$$

We set

$$\alpha \cong \beta \iff \alpha \text{ and } \beta \text{ are naturally isomorphic.}$$

§3.21. **Referential synonymy**. Two proper terms are *referentially synonymous* if they have naturally isomorphic referential intensions, and two immediate terms are referentially synonymous if they have the same denotations. In symbols:

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

The Referential Synonymy Theorem §3.4 follows immediately from this definition, and it is much easier to understand and apply than chasing natural isomorphisms. For the purposes of this paper, it might as well be taken as the definition of referential synonymy.

Together with the rules for reduction, the Referential Synonymy Theorem implies easily the rules for referential synonymy in Table 8. Notice the last rule, which is a very weak form of $\beta$-reduction—and just about the only form of $\beta$-reduction which is valid for referential synonymy. For a simple counterexample to a minimal,

plausible strengthening, let's appeal once more to John's self-love. Easily from the rules:

$$\text{John loves himself} \xrightarrow{\text{render}} \Big(\lambda(x)\mathsf{loves}(x,x)\Big)(\mathsf{John})$$
$$\Rightarrow_{\mathrm{cf}} \Big(\lambda(x)\mathsf{loves}(x,x)\Big)(j) \text{ where } \{j := \mathsf{John}\},$$
$$\text{John loves John} \xrightarrow{\text{render}} \mathsf{loves}(\mathsf{John},\mathsf{John})$$
$$\Rightarrow_{\mathrm{cf}} \mathsf{loves}(j_1,j_2) \text{ where } \{j_1 := \mathsf{John}, j_2 := \mathsf{John}\},$$

and so[28]

$$\Big(\lambda(x)\mathsf{loves}(x,x)\Big)(\mathsf{John}) \not\approx \mathsf{loves}(\mathsf{John},\mathsf{John}),$$

since (for one thing) their referential intensions have different dimensions.

From the conceptual point of view, it might be better to define referential synonymy only between proper terms, since the words suggest "same meaning" and immediate terms are not assigned meanings. The rules in Table 8 are simpler and easier to use, however, when we have the relation defined between pairs of arbitrary terms, for example in the proof of the

§3.22. **Compositionality Theorem**. *For all terms $A$, $B$, $C$ and every variable $x$ such that* $\mathrm{type}(x) = \mathrm{type}(B) = \mathrm{type}(C)$,

$$\textit{if } B \approx C, \textit{ then } A\{x :\equiv B\} \approx A\{x :\equiv C\},$$

*assuming that the substitutions are free.*

PROOF is by induction on the term $A$, applying the rules in Table 8. ⊣

§3.23. **Why not assign meanings to immediate terms?** Especially since they have canonical forms, which define (trivial) acyclic recursors, and so there is an obvious candidate for the object $\mathrm{int}(x)$.

Suppose our structure has a constant $\mathsf{id} : \mathsf{e} \to \mathsf{e}$ for the identify function on the set of entities,

$$\mathrm{id}(x) = x \quad (x \in \mathbb{T}_{\mathsf{e}}),$$

so that $\mathsf{id}(x)$ and $x$ are both irreducible and denotationally equivalent, and they would be synonymous under any plausible assignment of meaning to variables which is consistent with the referential intensions approach; but then compositionality would fail, since

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

and so $\mathsf{f}(\mathsf{id}(x)) \not\approx \mathsf{f}(x)$. In this calculus of referential intensions, variables (and the more general, immediate terms) behave a little like 0 in the arithmetic of fractions:

[28] In fact we cannot strengthen the rule to allow a location $q$ in place of the pure variable $u$, because, with a constant $\mathsf{f}$,

$$\Big(\lambda(v)\mathsf{f}(p(v))\Big)(q) \not\approx \mathsf{f}(p(q)) \Rightarrow_{\mathrm{cf}} \mathsf{f}(r) \text{ where } \{r := p(q)\},$$

since the term $\Big(\lambda(v)\mathsf{f}(p(v))\Big)(q)$ on the left is irreducible. (Incidentally, the example suggests that explicit, irreducible terms can be quite complex.)

it is simply not possible to assign any conventional (trivial) value to $\frac{1}{0}$ and still have the usual rules of arithmetic hold.

The Reduction and Synonymy Calculi are very effective tools for establishing simple synonymies; for example,

$$(A = B) \approx (B = A) \qquad (\text{type}(A) = \text{type}(B)),$$

since

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

where the crucial, second step is valid because

$$\models a = b \iff b = a.$$

Proofs of non-synonymy are not so simple when complex terms are involved, because it is tedious to compute canonical forms. We show first that acyclic recursion produces more meanings than can be expressed in the typed $\lambda$-calculus.

§3.24. **Theorem**. (1) *No location occurs in more than one part of an explicit term*.
(2) *Suppose a location $p$ occurs in two parts $A_k$ and $A_l$ of a term $A$, and neither $A_k$ nor $A_l$ denotes a function which is independent of $p$, i.e., for some assignment to the variables* g *and objects $r, r'$,*

$$\text{den}(A_k)(\text{g}\{p := r\}) \neq \text{den}(A_k)(\text{g}\{p := r'\}),$$

*and similarly with $A_l$. It follows that $A$ is not referentially synonymous with any explicit term*.

PROOF. (1) is very easy, by induction on the definition of explicit terms and using the rules (CF1)–(CF4) in the construction of canonical forms §3.13. For example, looking at (CF4), the only way in which some $p'_i$ can occur in $A'_k$ and also in $A'_l$, with $k \neq l$, is if $p_i$ occurred in both $A_k$ and $A_l$, which is ruled out by the induction hypothesis.

For (2), assume the hypothesis and (towards a contradiction) that $A \approx B$ with an explicit $B$, so that

$$\begin{aligned} A &\Rightarrow_{\text{cf}} A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\}, \\ B &\Rightarrow_{\text{cf}} B_0 \text{ where } \{p_1 := B_1, \ldots, p_n := B_n\}, \end{aligned}$$

with the denotations matching, and in particular, for all g,

$$\text{den}(A_k)(\text{g}) = \text{den}(B_k)(\text{g}), \quad \text{den}(A_l)(\text{g}) = \text{den}(B_l)(\text{g}).$$

By the hypothesis then, there is a g and $r, r'$ such that

$$\text{den}(B_k)(\text{g}\{p := r\}) \neq \text{den}(B_k)(\text{g}\{p := r'\}),$$

and this is not possible unless $p$ occurs in $B_k$; and by the same argument, $p$ must also occur in $B_l$, which contradicts (1). ⊣

§3.25. **Corollary**. *The terms in* (16) *and* (17) *are not referentially synonymous with explicit terms.*

PROOF. If wife is a constant, then the canonical form of (16) is

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

and the canonical form of (17) is (18); the location $j$ occurs in at least two of the parts of each of these canonical forms, and so Theorem §3.24 applies. If wife is a closed term, e.g.,

$$\mathsf{wife} \equiv \lambda(u)\Big(\mathsf{the}(\lambda(v)\mathsf{married}(u, v))\Big),$$

then $j$ will occur in the subsequent reduction of $\mathsf{wife}(j)$ to canonical form, and we can again apply the theorem. ⊣

§3.26. **Co-indexing, coordination and logical form**. It can be argued that the "new meanings" of $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$ which cannot be expressed by explicit terms are not mere curiosities. Consider the following two, related sentences, assuming for simplicity that stumbled and fell are constants:

(28) John stumbled and he fell (co-indexing)

(29) John stumbled and fell (coordination)

Their rendering requires co-indexing and coordination as indicated, and if we perform these operations using abstraction in the most natural way (as in §2.8 and §2.10), we get exactly the same formal term:

$$\text{John stumbled and he fell} \xrightarrow{\text{render}}_{\lambda} \lambda(x)\Big(\mathsf{stumbled}(x)\,\&\,\mathsf{fell}(x)\Big)(\mathsf{John})$$

$$\text{John stumbled and fell} \xrightarrow{\text{render}}_{\lambda} \lambda(x)\Big(\mathsf{stumbled}(x)\,\&\,\mathsf{fell}(x)\Big)(\mathsf{John})$$

This is surely wrong, as it implies that the two sentences (28) and (29) have the same logical form, which evidently they do not: (28) is a conjunction, while (29) is a predication. If we do the co-indexing and the coordination using the recursion construct (following §2.8 and §2.10 again), we get instead

(30) John stumbled and he fell $\xrightarrow{\text{render}}_{\mathrm{ar}} \mathsf{stumbled}(j)\,\&\,\mathsf{fell}(j) \text{ where } \{j := \mathsf{John}\}$,

(31) John stumbled and fell

$$\xrightarrow{\text{render}}_{\mathrm{ar}} \Big(\lambda(x)(s(x)\,\&\,f(x)) \text{ where } \{s := \mathsf{stumbled},\ f := \mathsf{fell}\}\Big)(\mathsf{John})$$

$$\Rightarrow_{\mathrm{cf}} \lambda(x)(s(x)\,\&\,f(x))(j) \text{ where } \{s := \mathsf{stumbled},\ f := \mathsf{fell},\ j := \mathsf{John}\}$$

It is clear from the indicated canonical forms of these two terms that they are not synonymous and they render correctly (28) as a conjunction and (29) as a predication. In fact, easily,

$$\lambda(x)\Big(\mathsf{stumbled}(x)\,\&\,\mathsf{fell}(x)\Big)(\mathsf{John})$$

$$\approx \lambda(x)(s(x)\,\&\,f(x))(j) \text{ where } \{s := \mathsf{stumbled},\ f := \mathsf{fell},\ j := \mathsf{John}\}$$

so that the explicit rendering captures coordination correctly, in this example, while it misses on the co-indexing: the formal term in (30) is not synonymous with any explicit term. This is an argument within referential intension theory, of course,

but the (apparent) identification of the explicit renderings of (28) and (29) suggests that rendering in LIL *produces the wrong logical forms*, and so they cannot serve as representations of meaning in any theory which derives meaning from logical form.

§3.27. **The symmetry of identity statements**. Let us also keep the promise made in §2.9, to show that with the Montague (quantifier) renderings, the identity statement "the evening star is the morning star" is not referentially synonymous with its converse, i.e.,

$$(32)\qquad \mathsf{ES}_{\mathrm{Mont}}(\lambda(u)\mathsf{MS}_{\mathrm{Mont}}(\lambda(v)(u=v))) \not\approx \mathsf{MS}_{\mathrm{Mont}}(\lambda(u)\mathsf{ES}_{\mathrm{Mont}}(\lambda(v)(u=v)))$$

We assume that

$$\mathsf{ES}_{\mathrm{Mont}} \equiv \mathsf{the}_{\mathrm{Mont}}\Big(\mathsf{first}(\mathsf{evening}(\mathsf{star}))\Big)$$
$$\mathsf{MS}_{\mathrm{Mont}} \equiv \mathsf{the}_{\mathrm{Mont}}\Big(\mathsf{last}(\mathsf{morning}(\mathsf{star}))\Big),$$

where $\mathsf{first}, \mathsf{evening}, \mathsf{last}, \mathsf{morning} : (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}})$ are adjectives, such that, for example, if $p$ is the property of being a "star" visible in the evening sky and $x$ is a "star", then

$$\mathrm{first}(p)(x) \iff x \text{ is visible before any other evening star.}$$

Moreover, the Montague description operator "$\mathsf{the}_{\mathrm{Mont}}$" acts on relations and produces quantifiers, i.e.,

$$\mathsf{the}_{\mathrm{Mont}} : (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{q}}.$$

We will also assume that $\mathsf{the}_{\mathrm{Mont}}$ is a constant of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ denoting this operator, but the other relevant terms (first, evening, ...) may be complex, and it is this which makes the non-synonymy argument a bit tedious.

PROOF of (32). Assume the opposite, and also, for simplicity, at first, that first and last are constants. Easily,

$$\begin{aligned}&\mathsf{ES}_{\mathrm{Mont}}(\lambda(u)\mathsf{MS}_{\mathrm{Mont}}(\lambda(v)(u=v)))\\ &\qquad\Rightarrow \mathsf{the}_{\mathrm{Mont}}(h)(r) \text{ where } \{h := \mathsf{first}(es), es := \mathsf{evening}(\mathsf{star}),\\ &\qquad\qquad r := \lambda(u)\mathsf{MS}_{\mathrm{Mont}}(\lambda(v)(u=v))\},\\ &\mathsf{MS}_{\mathrm{Mont}}(\lambda(u)\mathsf{ES}_{\mathrm{Mont}}(\lambda(v)(u=v)))\\ &\qquad\Rightarrow \mathsf{the}_{\mathrm{Mont}}(p)(r) \text{ where } \{p := \mathsf{last}(ms), ms := \mathsf{morning}(\mathsf{star})\\ &\qquad\qquad r := \lambda(u)\mathsf{ES}_{\mathrm{Mont}}(\lambda(v)(u=v))\}.\end{aligned}$$

The subsequent reduction of these terms to canonical form will not affect their heads and the assignments to $h$ and $p$ which are already explicit and irreducible, and so by Theorem §3.4 we will have

$$\begin{aligned}&\mathsf{ES}_{\mathrm{Mont}}(\lambda(u)\mathsf{MS}_{\mathrm{Mont}}(\lambda(v)(u=v)))\\ &\qquad\Rightarrow_{\mathrm{cf}} \mathsf{the}_{\mathrm{Mont}}(p_i) \text{ where } \{p_i := \mathsf{first}(p_j), p_1 := A_1, \ldots, p_s := A_s\},\\ &\mathsf{MS}_{\mathrm{Mont}}(\lambda(u)\mathsf{ES}_{\mathrm{Mont}}(\lambda(v)(u=v)))\\ &\qquad\Rightarrow \mathsf{the}_{\mathrm{Mont}}(p_k) \text{ where } \{p_k := \mathsf{last}(p_l), p_1 := B_1, \ldots, p_s := B_s\},\end{aligned}$$

where

$$\models \mathsf{the}_{\mathrm{Mont}}(p_i) = \mathsf{the}_{\mathrm{Mont}}(p_k),\ \models \mathsf{first}(p_j) = \mathsf{last}(p_l),\ \models A_m = B_m \quad (m = 1, \ldots, s).$$

Now

$$\not\models \mathsf{the}_{\mathrm{Mont}}(p_i) = \mathsf{the}_{\mathrm{Mont}}(p_k)$$

if the locations $p_i$ and $p_k$ are distinct, simply because the operator $\mathsf{the}_{\mathrm{Mont}}$ is not constant; and so we must have that $p_i \equiv p_k$. Thus

$$\models \mathsf{first}(p_j) = \mathsf{last}(p_l),$$

which is absurd, whether the locations $p_j$ and $p_l$ are distinct or identical.

The argument is just a bit more tedious if first and last are complex terms. ⊣

**§4. Local meanings and demonstratives.** In this section we will consider two standard puzzles about substitutivity in the Philosophy of Language. Both of these have been introduced in the literature as puzzles about *belief*, but all they assume about it is a principle already found in Frege: that *if a rational person can reasonably believe A and not believe B in the same context* (state)*, then A and B are not synonymous*. Thus they are really puzzles about synonymy, and it is easy to formulate them precisely within the theory of referential intensions and see what (if anything) it has to say about them—and they about it.

At the end of the section we will also make a brief comment about the relevance of referential intensions to the understanding of "potential knowledge".

§4.1. **Dependence of belief statements on the state**. I believe now that *9931 is a prime number*, but I did not believe it last year, although "9931 is a prime number" certainly meant the same then as it does now; it is my belief system which changed, after I did some computations. On the other hand, I also believe now that *John loves Mary* and I did not believe it last year, although my beliefs about love have not changed; it is just that "John loves Mary" meant something quite different then—it was, in fact, false, as John first met Mary in January. Thus the state affects belief statements in two independent ways, as our system of beliefs but also the meaning of sentences depend on it. It is useful to separate these two effects, and take as the objects of belief the *situated* (local) meanings of Carnap intensions.

§4.2. An **utterance**[29] is a pair $(A, a)$ of a closed Carnap intension $A : \tilde{\mathsf{t}}$ and a state $a$. To deal effectively with these quasi-syntactic objects, it is useful to add to the language $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ a *parameter* $\overline{a}$ for each state $a$, so that we can identify an utterance $(A, a)$ with the term $A(\overline{a}) : \mathsf{t}$. These state parameters are not constants, and from the syntactic point of view they behave exactly like pure variables of type $\mathsf{s}$ for which the value of every assignment has been fixed. For example,

$$\mathsf{loves}(\mathsf{John}, \mathsf{Mary})(\overline{a}) \Rightarrow_{\mathrm{cf}} \mathsf{loves}(j, m)(\overline{a}) \text{ where } \{j := \mathsf{John}, m := \mathsf{Mary}\},$$

and the term on the right is the canonical form of the utterance on the left because it is irreducible—which it would not be if $\overline{a}$ were a constant.

[29] Perhaps a misnomer, the term is chosen because one of the basic and most puzzling functions of the state is to specify the speaker ("I"), the time ("now"), etc.

Notice that once we add state parameters to the language, they can occur anywhere in a term, like variables; but by "utterance" we will always mean a term of the form $A(\overline{a})$, where $A$ is a closed and parameter-free Carnap intension.

The referential intension $\mathrm{int}(A(\overline{a}))$ models the *local meaning* of the Carnap intension $A$ *in state* $a$. This function is very simple in the present theory; because if $A$ is closed and

$$A \Rightarrow_{\mathrm{cf}} A_0 \text{ where } \{p_1 := A_1, \dots, p_n := A_n\} : \tilde{\mathsf{t}},$$

then by the recap rule,

$$A(\overline{a}) \Rightarrow_{\mathrm{cf}} A_0(\overline{a}) \text{ where } \{p_1 := A_1, \dots, p_n := A_n\} : \mathsf{t},$$

so that the parameter $\overline{a}$ occurs only in the head part of $A(\overline{a})$. Thus $\mathrm{int}(A(\overline{a}))$ is obtained from $\mathrm{int}(A)$ by leaving the body of the recursor untouched and simply applying its head function to the state $a$.

If $A(\overline{a}) \approx B(\overline{a})$, we say that $A$ and $B$ are (locally) *synonymous in state* $a$.

With these notions in place, we can now formulate a much-simplified, monolingual version of the *Pierre puzzle* in Kripke [1979].

§4.3. **Los Angeles**. *Petros emigrated from his native Greece to the United States at a rather advanced age, and immediately fell in love with the city of Los Angeles, where he settled. Every chance he gets he declares proudly:*

(A) I live in Los Angeles.

*When, however, a new acquaintance who had heard of this tried to start conversation with an innocent* "I hear you live in LA"*, Petros looked puzzled, declared again that he lives in Los Angeles, and added emphatically*:

(B) I do not live in LA.

Let us now stipulate that the language has constants

$$\mathsf{Los\ Angeles}, \mathsf{LA} : \tilde{\mathsf{e}}$$

which refer rigidly to the same largest city in California, so that

(33) $$\models \mathsf{Los\ Angeles} = \mathsf{LA}.$$

This is reasonable, as both abbreviations of the full name of Los Angeles are well established,[30] and it implies that

(34) $$\mathsf{Los\ Angeles} \approx \mathsf{LA},$$

since Los Angeles and LA are explicit and irreducible. The Compositionality Theorem §3.22 now yields

(35) $$\mathsf{reside}(\mathsf{I}, \mathsf{Los\ Angeles})(\overline{a}) \approx \mathsf{reside}(\mathsf{I}, \mathsf{LA})(\overline{a})$$

for the state $a$ of Petros' two utterances, whatever term (or constant) renders the residence relation. Thus Petros appears to (rationally) believe one utterance and disbelieve a synonymous one, which contradicts our taking utterances as the carriers of belief.

[30] The full name of Los Angeles is *El Pueblo de Nuestra Señora, la Reyna de Los Angeles de Porciúncula*.

The common-sense resolution of the puzzle was expressed by Petros' sister Maria, who commented to those present when her brother made his remarks,

(36) He doesn't know that Los Angeles is LA.

Now Kripke argues, correctly, that this does not amount to an explanation, because (with the assumptions we have made), Maria's comment is synonymous with

He doesn't know that Los Angeles is Los Angeles,

which robs it of its explanatory power, and is probably false. So, still following Kripke, we have a genuine puzzle, which means that the example fails to satisfy one of our basic assumptions about semantics; and the most likely culprit, in this case, seems to be Frege's famous doctrine about knowledge of the language:

> The sense of a proper name is grasped by everybody who is sufficiently familiar with the language or totality of designations to which it belongs ... Comprehensive knowledge of the thing denoted ... we never attain (Frege [1892], 27]).

To resolve the puzzle, we must argue that someone "sufficiently familiar with the language" in Frege's sense cannot (rationally) utter in the same state both (A) and (B), in other words, that Petros is not a language speaker—he is incoherent.

§4.4. **Language speakers**. There are probably no English speakers who satisfy Frege's stringent criterion of "grasping the totality of designations" of the language. "The language speaker" is an idealization, which we assume in order to develop a logical theory of meaning, much as we assume the existence of perfect vacuum and complete absence of friction in order to develop a mathematical theory of Newtonian mechanics. It is not an internal matter of logic, and its utility must be judged by the plausibility of the conclusions derived from it together with the other (also idealized) hypotheses of the theory. Nevertheless, it is worth examining exactly how much of Frege's doctrine about language speakers we need to accept, and trying to formulate it in logical rather than metaphysical terms.

What does it mean *to grasp* the sense of a linguistic expression? It is generally assumed that Frege understood senses to be abstract objects, functions and the like, and this already leads to classical metaphysical questions: how do we "grasp" 0, or the notion of natural number? Moreover, unlike Frege, we have allowed constants which refer directly and rigidly to objects that are definitely not abstract, like Los Angeles, and this complicates the problem: part of the referential intension of $\mathsf{reside}(\mathsf{I}, \mathsf{Los\ Angeles})(\overline{a})$ is the constant function with value Los Angeles, i.e., essentially, Los Angeles (the object), and I have no idea what it means to grasp it. It is good to replace metaphysical hypotheses of this type by assumptions which can be formulated in logical terms. The key to this is Maria's explanation (36), if we understand it not within the language, but as a metalinguistic claim about Petros' insufficient knowledge of the language, i.e., in the form

He doesn't know that "LA" is another name for Los Angeles.

In short, Petros is incoherent not because he cannot "grasp Los Angeles" (which may not be possible), but because he does not know the crucial, denotational

identity (33), which implies the synonymy (34). Thus, what we need to assume of a language speaker is that (at a minimum) *he knows all true identities*

$$\models \mathsf{a} = \mathsf{b} \tag{37}$$

between constants of the language, of any type. This is a tall order, to be sure, and Petros fails it, but it is a much easier test to make precise (and pass) than Frege's demand about "grasping". After some forty years of living in Los Angeles, I still make no claim that I can "grasp it" (whatever that means), but I certainly know that

$$\models \mathsf{Los\ Angeles} = \mathsf{LA},$$

and I use both of these names interchangeably, often with no recollection of which one I employed in any particular utterance.

§4.5. **Is referential synonymy decidable**? Unfortunately, a full-bodied logical version of Frege's doctrine about "grasping senses" demands more of the language speaker than the knowledge of all identities between constants as in (37). For example, with the most natural assumption about the meaning of "between", easily

(38) Los Angeles is between the desert and the sea

$$\approx \text{Los Angeles is between the sea and the desert,}$$

and the language speaker should recognize this synonymy along with all other synonymies. In particular, if we do not want to endow the language speaker with truly supernatural abilities, doing full logical justice to Frege's "grasping doctrine" requires proof of the following purely technical

**Main Conjecture**. *If the set of constants $K$ is finite, then the relation of referential synonymy between closed terms of* $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$ *is decidable*.[31]

This is still open, although it was shown in Moschovakis [1994] for the language FLR, which extends a reasonably large fragment of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$.[32] By the Referential Synonymy Theorem §3.4, the Main Conjecture is equivalent to the decidability of denotational identities of the form

$$\models A = B$$

between *explicit irreducible terms* $A$, $B$. For example, if we assume for simplicity that "between" is a constant, then (38) follows by compositionality from

$$\models \mathsf{between}(x, y, z) = \mathsf{between}(z, y, x),$$

which is a denotational identity between explicit, irreducible terms. The language has, however, many and complex explicit, irreducible terms, and so the proof of the full conjecture is not all here.

For a satisfactory development of a theory of belief in which the belief carriers are utterances, we would also need to establish the decidability of synonymy between the parts of utterances in which the parameter $\overline{a}$ occurs. The question is not so simple to make precise, and we will leave it for Kalyvianaki and Moschovakis [].

[31] There is some evidence that Frege believed some version of the Main Conjecture, on the basis of a 1906 letter to Husserl, see Heijenoort [1985].

[32] There is unfortunately a gap in the proof given in Moschovakis [1994], but it can be easily filled and the result is correct.

Finally, it should be pointed out that although a proof of the Main Conjecture is highly desirable, the status of the conjecture does not affect the development of referential intension theory or its possible applicability to computational semantics. If the Main Conjecture is false, well, then no human being can be a language speaker in principle—but then we already know that, in practice, there are no (perfect) language speakers.

We now turn to the second puzzle, which was introduced by Salmon and Soames in the introduction to Church [1962] and which is worth quoting verbatim.[33]

§4.6. **Is he Scott**? *Let us suppose that in a book-signing ceremony given by "the author of Waverley", a cleverly disguised Scott autographs King George's copy of Waverley. King George, being fooled by Scott's disguise, concludes that Waverley was written by someone other than Scott. He sincerely declares*

(D) He is not Scott

*pointing at the disguised author. Yet King George surely disbelieves, and would vigorously deny that*

(E) Scott is not Scott.

Now the puzzle comes from the circumstance that

$$\mathsf{He}(a) = \mathsf{Scott}(a) \tag{39}$$

for the state $a$ at the book-signing, which implies immediately that

$$\mathsf{He}(\overline{a}) \approx \mathsf{Scott}(\overline{a}),$$

since these two terms are explicit and irreducible. One might suspect from this that (skipping the irrelevant negations)

$$\Big(\text{He is Scott}\Big)(\overline{a}) \approx \Big(\text{Scott is Scott}\Big)(\overline{a}), \quad \text{(Caution: this is false!)} \tag{40}$$

and that would make King George guilty of incoherence. But (40) is not true:

$$\begin{aligned}\Big(\text{He is Scott}\Big)(\overline{a}) &\xrightarrow{\text{render}} \Big(\mathsf{He} = \mathsf{Scott}\Big)(\overline{a}) \\ &\Rightarrow_{\text{cf}} (h = s)(\overline{a}) \text{ where } \{h := \mathsf{He}, s := \mathsf{Scott}\} \\ &\approx h(\overline{a}) = s(\overline{a}) \text{ where } \{h := \mathsf{He}, s := \mathsf{Scott}\},\end{aligned} \tag{41}$$

$$\begin{aligned}\Big(\text{Scott is Scott}\Big)(\overline{a}) &\xrightarrow{\text{render}} \Big(\mathsf{Scott} = \mathsf{Scott}\Big)(\overline{a}) \\ &\Rightarrow_{\text{cf}} (s' = s)(\overline{a}) \text{ where } \{s' := \mathsf{Scott}, s := \mathsf{Scott}\} \\ &\approx s'(\overline{a}) = s(\overline{a}) \text{ where } \{s' := \mathsf{Scott}, s := \mathsf{Scott}\},\end{aligned} \tag{42}$$

and by the Referential Synonymy Theorem §3.4

$$\Big(\mathsf{He} = \mathsf{Scott}\Big)(\overline{a}) \not\approx \Big(\mathsf{Scott} = \mathsf{Scott}\Big)(\overline{a}), \tag{43}$$

simply because

$$\models \mathsf{He} \neq \mathsf{Scott}.$$

[33] Salmon and Soames started from a related puzzle of Church [1962], which is also about belief but employs variables rather than descriptions or demonstratives.

So George IV makes two non-synonymous utterances, one false one true; he may be muddled, but he is not incoherent.

§4.7. **Individual concepts in utterances**. The good King can hold onto his erroneous belief that the man who autographed his book is not himself, while poor Petros is not allowed to believe falsely that he does not live where he lives, on pain of incoherence. This is because, intuitively: *if you mention an individual concept, then that (full) concept is part of the meaning of your utterance*.[34] In the two puzzles above, Los Angeles, LA, He and Scott are all parts of the relevant terms, but Los Angeles = LA, which dooms poor Petros, while He $\neq$ Scott, which saves the King.

We have already discussed in §4.2 the technical fact behind this claim: *the state parameter $\overline{a}$ occurs only in the head of the canonical form of an utterance $A(\overline{a})$ and not in its body*.

In some more detail, the head of an utterance must have type $\mathsf{t}$, and so it cannot be $\mathsf{c}(\overline{a})$ for any constant $\mathsf{c} : \tilde{\mathsf{e}}$ denoting an individual concept; hence every such constant which occurs in a closed Carnap intension $A : \tilde{\mathsf{t}}$ is a part of the canonical form of every utterance $A(\overline{a})$ of $A$, and every utterance synonymous with $A(\overline{a})$ must contain some constant synonymous with $\mathsf{c}$.

§4.8. **Impossible utterances**. Technical explanations are not very satisfying: we are left with the feeling that, whatever the technicalities, the King intended to say that he does not believe

$$\mathsf{He}(\overline{a}) = \mathsf{Scott}(\overline{a}), \tag{44}$$

which seems to mean exactly the same as

$$\mathsf{Scott}(\overline{a}) = \mathsf{Scott}(\overline{a}). \tag{45}$$

Well, if the King had actually denied (44), then, indeed, we would have had a puzzle, because by compositionality and (39),

$$\mathsf{He}(\overline{a}) = \mathsf{Scott}(\overline{a}) \approx \mathsf{Scott}(\overline{a}) = \mathsf{Scott}(\overline{a}). \tag{46}$$

But the King did not deny (44), and, indeed, *he could not have denied* (44) *because* (44) *is not an utterance*.[35] It is a term of type $\mathsf{t}$, to be sure, if we take "=" to be the equality relation on $\mathbb{T}_{\mathsf{e}}$, but it does not have the logical form $A(\overline{a})$ of an utterance. The basic principle here is that *the only syntactic expressions we can affirm or deny are closed Carnap intensions*, which are interpreted in the current state to produce an utterance; we cannot use the parameter naming the current (or any other) state, any more than we can use a free variable when we speak.

[34] Russell would put the city of Los Angeles in the proposition expressed by Petros' utterance, while the referential intension of that utterance contains (as a part) the constant function which assigns the city to every state; there is little difference between the two.

[35] It may be argued that (by the Gallin interpretation), (44) is exactly what the King denies when his utterance "He is not Scott" is rendered in LIL and $\overline{a}$ is the current state. We will discuss this in Kalyvianaki and Moschovakis [] and claim that as renderings of utterances in LIL are naturally understood, they cannot serve as belief carriers—although they express a robust notion of *information content*, related to but different from local meaning.

Suppose we try to correct this deficiency of the language by introducing a constant book-signing : $\tilde{\mathsf{s}}$ which denotes rigidly the relevant state,

$\mathrm{den}(\text{book-signing})(a) =$ the state in which the book-signing ceremony took place.

If we replace the state parameter $\overline{a}$ by the constant book-signing in the terms of (46), we get

$$\mathsf{He}(\text{book-signing})(\overline{a}) = \mathsf{Scott}(\text{book-signing})(\overline{a}), \tag{47}$$

$$\mathsf{Scott}(\text{book-signing})(\overline{a}) = \mathsf{Scott}(\text{book-signing})(\overline{a}), \tag{48}$$

and the King can try to deny the first while asserting the second. But the constant He is a part of (47) and not (denotationally) equal to any part of (48), and so these two utterances are not synonymous[36] and the good King has once more escaped incoherence.

To summarize the discussion in this section, what we inferred from the Kripke example was that puzzles which are grounded on a *lack of knowledge of the language* are not relevant to the development of Fregean semantics, which assume from the get go that the "language speakers" know the language perfectly; and we suggested that the Salmon-Soames puzzle is based on a confusion of the utterance $(\mathsf{He\ is\ Scott})(\overline{a})$ with the closed term $\mathsf{He}(\overline{a}) = \mathsf{Scott}(\overline{a})$, which means something entirely different—and is not an utterance.

§4.9. **Potential knowledge**. Montague—and practically everybody else—admits a modal interpretation of knowledge: we assume a constant $\mathsf{K}_i : \tilde{\mathsf{t}} \to \tilde{\mathsf{t}}$ for each "agent" $i$, and we interpret it by

$$K_i(p, a) \iff (\forall b \in \mathbb{T}_{\mathsf{s}})[D_i(a, b) \Longrightarrow p(b)], \tag{49}$$

where, intuitively,

$$D_i(b, a) \iff \text{the state } b \text{ is accessible to } i \text{ from the state } a.$$

I have dismissed (with many others) this understanding of propositional attitudes because of the *problem of omniscience*: by it, if you know one true, mathematical statement (like $1 + 1 = 2$), you know them all. It seems to me that customary knowledge of the truth of an utterance $A(\overline{a})$ is grounded on the meaning of $A(\overline{a})$, i.e., on the referential intension $\mathrm{int}(A)(\overline{a})$ by the present modeling of meanings; it is only that (unfortunately), we do not know now how to define properly the meaning of knowledge claims, i.e., the operation

$$(A, a) \mapsto \mathrm{int}(K_i A(\overline{a})).$$

On the other hand, if by $\mathsf{K}_i A(\overline{a})$ we understand that the agent $i$ has *potential knowledge* of $A(\overline{a})$—she has access to the relevant facts, from which she could deduce the truth or falsity of $A(\overline{a})$, if only she were smart enough—then the Montague

[36] The non-synonymy becomes more obvious if we look at the canonical forms of these two terms:

$\mathsf{He}(\text{book-signing})(\overline{a}) = \mathsf{Scott}(\text{book-signing})(\overline{a})$

$\Rightarrow_{\mathrm{cf}} (a = b)(\overline{a})$ where $\{a := \mathsf{He}(h_1), h_1 := \text{book-signing}, b := \mathsf{Scott}(h_2), h_2 := \text{book-signing}\}$,

$\mathsf{Scott}(\text{book-signing}) = \mathsf{Scott}(\text{book-signing})(\overline{a})$

$\Rightarrow_{\mathrm{cf}} (a = b)(\overline{a})$ where $\{a := \mathsf{Scott}(h_1), h_1 := \text{book-signing}, b := \mathsf{Scott}(h_2), h_2 := \text{book-signing}\}$.

interpretation makes perfect sense; and there is no doubt that this notion of potential knowledge is a natural and useful one, especially in the analysis of the behavior of computing systems. The only (minor) point perhaps worth making here is that the referential theory of meaning can incorporate potential knowledge and add something to it: the reduction

$$\mathsf{K}_i A \Rightarrow \mathsf{K}_i(p) \text{ where } \{p := A\}$$

provides a meaning (not just a truth value) to the potential knowledge claim, so that "knowing that there are infinitely many prime numbers" at least means something different from "knowing that $1 + 1 = 2$", even though the two potential knowledge claims are denotationally equivalent.

**§5. English as a programming language.** The starting point for the work reported in this article, was the insight that a correct understanding of programming languages should explain the relation between a program and the algorithm it expresses, so that the basic interpretation scheme for a programming language is of the form

$$\text{program } P \mapsto \text{algorithm}(P) \mapsto \text{den}(P). \tag{50}$$

It is not hard to work out the mathematical theory of a suitably abstract notion of algorithm which makes this work; and once this is done, then it is hard to miss the similarity of (50) with the basic Fregean scheme for the interpretation of a natural language,

$$\text{term } A \mapsto \text{meaning}(A) \mapsto \text{den}(A). \tag{51}$$

This suggested at least a formal analogy between algorithms and meanings which seemed worth investigating, and proved after some work to be more than formal: when we view natural language with a programmer's eye, it seems almost obvious that we can represent the meaning of a term $A$ by the algorithm which is expressed by $A$ and which computes its denotation. This is the view which I have tried to explain and apply in this article.

Aside from the relation between algorithms and meanings, programming languages resemble natural languages more than they resemble the classical, formal languages of logic, both in their complexity and also because they exhibit some natural language phenomena which are absent from formal languages.[37] These ideas are well known and understood, I have used them in the main part of the article, and I will not discuss them further here. I will also not try to explain my take on basic philosophical questions like what it means to "define", "represent faithfully" or "explicate" *meaning* (or any other notion) in set-theoretic terms; I tried my best to be as clear on these issues as I can in Moschovakis [1998], and it is unlikely that I can improve on it in this brief section. So I will confine myself here to a few, general remarks on the semantics of programming languages, and to motivating the specific choice of "abstract algorithms" which can justify the scheme (50).

§5.1. **Denotational semantics for programming languages**. These were introduced in Scott and Strachey [1971], and they have been developed very extensively since

[37]The distinction between "lexical" and "dynamic" scoping in Lisp, for example, is very much like (if not identical to) that between the de dicto and de re readings of sentences which involve demonstratives, i.e., Kaplan's main problem in Kaplan [1978b].

then by Scott and his students and followers. Scott's aim was to construct Fregean, compositional semantics of denotations for programming languages: he showed how to assign a semantic value to each syntactically correct part of a programming languages by "structural recursion" on the syntax, so that ultimately each program is assigned a denotation. The theory is mathematically challenging (and correspondingly interesting), not only because the syntax of programming languages is complex, but also because what programs denote is not always simple: sometimes it is just a number or a function, but it can also be a sequence of "acts" (printouts, for example, or pictures on a screen) or an "interactive behavior"—and to make the theory precise, these objects must be coded by appropriate, mathematical structures.

The development of denotational semantics represented a fundamental advance in the study of programming languages, both in our understanding of them and also in using them. Among other things, it made precise what it means for an *implementation* of a programming language to be correct: it must "execute" each program $P$ so that the "output" is the correct denotation of $P$, as predicted by the agreed-upon denotational semantics for the language.

At the same time, Scott's theory is peculiarly incomplete in that it makes no room for the notion of algorithm which (one would think) is at the heart of the matter. Consider, for example, the problem of "sorting" (putting in alphabetical order) a long list of words $u$. There are many algorithms which will do this—the *bubble sort*, the *merge sort*, the *quick sort* etc.— and they differ greatly in many ways, for example their *efficiency*. They can all be "programmed" (expressed) in every suffucently rich programming language $L$, but the denotational semantics of $L$ cannot distinguish between them, as they all have the same denotation, the function which assigns to each $u$ its alphabetized rearrangement. And so it seemed to me that Scott semantics should be refined by the introduction of algorithms as the primary semantics values of programs, which then determine their denotations, i.e., by adopting the basic interpretation scheme (50).

In fact, it is not difficult to work out this refinement of Scott's theory, because almost all the technical tools required are already present in the mathematical theory of denotational semantics. The only subtle difficulty was the need to uncover the correct notion of algorithm, and I will turn to this next.

§5.2. **What is an algorithm?**[38] The classical Euclidean algorithm for the computation of the greatest common divisor of two natural numbers can be expressed succinctly by the recursive equation,

$$\begin{aligned}(52)\qquad \gcd(x,y) = \text{if } (\text{rem}(x,y)=0) \text{ then } y \\ \text{else } \gcd(y,\text{rem}(x,y)) \qquad (x \geq y \geq 1),\end{aligned}$$

where the *remainder* $\text{rem}(x,y)$ is the unique number $r$ such that for some $q$,

$$x = yq + r, \quad 0 \leq r < y;$$

and the gist of the view about algorithms defended in Moschovakis [1998] is that (52) *is all there is to the Euclidean algorithm*—this single equation specifies *what the*

[38]The arguments here are from Moschovakis [1998], which explains them in much more detail using the *mergesort* as the basic example.

*algorithm is*, and expresses it so that its fundamental properties can be most easily deduced. Consider the following.

**1. Implementations**. It is quite simple to verify that equation (52) is true of the greatest-common-divisor function,[39] but it does not look like the usual "instructions" and "commands" that we expect of an algorithm specification. Such instructions, however, are easy to extract from (52), if we read it as a "self-referential" definition of $\gcd(x, y)$. For example, let (recursively)

$$r_0 = x,\ r_1 = y,\ r_{n+2} = \mathrm{rem}(r_n, r_{n+1});$$

these "successive remainders" can be computed by applying (52) repeatedly, and

$$\text{if } k = \text{ the least } n \text{ such that } r_n = 0, \text{ then } \gcd(x, y) = r_{k-1}.$$

Or we might "re-write" (52) as a "while program",

$$X := x;\ Y := y; \text{while}(Y \neq 0)\Big(T := X; X := Y; Y := \mathrm{rem}(T, Y)\Big); \text{return } X.$$

It might appear that some understanding of "the Euclidean algorithm" beyond the information coded in (52) is required in order to extract a computational procedure from the equation, but this is not true: these extraction processes go by the fancy name of *implementations of recursion* and they can be automated—they are part of what a "compiler" or "interpreter" does for a sufficiently rich programming language (like Lisp or Pascal) whose syntax allows recursive definitions such as (52).

**2. Recursive equations vs. implementations**. Shouldn't the Euclidean "be" (or be represented by) one of the specific implementations just discussed rather than some other, more abstract object directly expressed by (52)? Well, the argument now is *which one*? In fact, there are many more implementations of the Euclidean than the two we mentioned, at least one for every programming language in which we can "program" (52) and for every processor which can execute the compiled version of this program. Which of these has a stronger claim "to be" the Euclidean? And what is common between all these different implementations which are (evidently) related by being "implementations of the Euclidean" if there is no single, abstract object which "is" the Euclidean?

**3. Properties of the Euclidean algorithm**. Let

$$d_\varepsilon(x, y) = \text{the number of divisions required to compute } \gcd(x, y) \text{ using (52)},$$

so that by reading (52) as a self-referential definition,

$$d_\varepsilon(x, y) = \text{if } (\mathrm{rem}(x, y) = 0) \text{ then } 1 \quad \text{else } 1 + d_\varepsilon(y, \mathrm{rem}(x, y)) \qquad (x \geq y \geq 1). \tag{53}$$

From this it follows by an easy induction on $y$[40] that

$$d_\varepsilon(x, y) \leq 2\log_2(y),$$

[39] It is true if $y$ divides $x$; and in the opposite case, when $\mathrm{rem}(x, y) > 0$, easily, for every $d$

$$[d \text{ divides } x \text{ and } d \text{ divides } y] \iff [d \text{ divides } y \text{ and } d \text{ divides } \mathrm{rem}(x, y)],$$

so that $\gcd(x, y) = \gcd(y, \mathrm{rem}(x, y))$.

[40] You need to consider three cases: whether $y$ divides $x$; otherwise, whether $\mathrm{rem}(x, y)$ divides $y$; and otherwise, whether $\mathrm{rem}(y, \mathrm{rem}(x, y))$ divides $\mathrm{rem}(x, y)$.

which is one version of the basic complexity estimate of the Euclidean. There are better versions of this and many other important properties of this algorithm, but they can all be deduced in similar ways directly from (52), without any specific reference to any specific way in which the Euclidean is implemented.

So it appears natural to identify the Euclidean algorithm with the "semantic content" of (52), and at first blush this can be none other than the function

$$f(x, y, p) = \text{if } (\text{rem}(x, y) = 0) \text{ then } y \text{ else } p(y, \text{rem}(x, y)), \tag{54}$$

where $p$ is a variable ranging over binary (partial) functions on the natural numbers. Notice first that this assumes the remainder operation and the conditional construct

$$C(u, s, t) = \text{if } (u = 0) \text{ then } s \text{ else } t$$

as "givens", not to be computed but to be "called". Moreover, (54) does not account for the "explicit computation" required to evaluate $f(x, y, p)$ for any given $x$, $y$ and $p$. Thus the final step is to break down the computation of $f(x, y, p)$ to its elementary steps—direct calls to the givens—and this is what (in the analogous case) the reduction calculus of this article adds to the process. When we do it here, we obtain the canonical form

$$\begin{aligned}\gcd(x, y) = p(x, y) \text{ where } \{p &:= \lambda(x)\lambda(y)C(q_1(x, y), y, r(x, y)),\\ q_1 &:= \lambda(x)\lambda(y)\text{rem}(x, y),\\ r &:= \lambda(x)\lambda(y)p(y, q_2(x, y)),\\ q_2 &:= \lambda(x)\lambda(y)\text{rem}(x, y)\}\end{aligned}$$

which leads to our representing the Euclidean by the *recursor*

$$\varepsilon = (\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4),$$

where, in effect (skipping the formal assignments),

$$\begin{aligned}\alpha_0(x, y, p, q_1, r, q_2) &= p(x, y),\\ \alpha_1(x, y, p, q_1, r, q_2) &= \lambda(x)\lambda(y)C(q_1(x, y), y, r(x, y)),\\ \alpha_2(x, y, p, q_1, r, q_2) &= \lambda(x)\lambda(y)\text{rem}(x, y) = \text{rem},\\ \alpha_3(x, y, p, q_1, r, q_2) &= \lambda(x)\lambda(y)p(y, q_2(x, y)),\\ \alpha_4(x, y, p, q_1, r, q_2) &= \lambda(x)\lambda(y)\text{rem}(x, y) = \text{rem}.\end{aligned}$$

It is an algorithm of the structure $(\mathbb{N}, C, \text{rem})$ where $\mathbb{N}$ is the set of natural numbers and $C$, rem are the conditional construct and the remainder function taken as "given".[41]

The same basic process identifies the algorithms of every mathematical structure $\mathfrak{A}$, comprising some given universes and certain operations on them taken as "given". Notice that these algorithms are always *relative to the givens*, and so they do not determine "absolutely computable" functions unless the givens are absolutely computable. For the case at hand, the process was somewhat more complex, because the intended interpretation of $\mathsf{L}^{\lambda}_{\text{ar}}(K)$ in §1.4 includes higher-type givens,

[41] And it is tempting here to co-index the two instances of rem, which will yield a minor (and not more efficient) variation of the Euclidean.

and somewhat simpler, because we did not need to consider truly recursive (self-referential) algorithms (like the Euclidean), and so the simpler, acyclic recursors sufficed.

## REFERENCES


Alonzo Church [1946], *A formulation of the logic of sense and denotation, abstract*, ***The Journal of Symbolic Logic***, vol. 11, p. 31.

Alonzo Church [1951a], *A formulation of the logic of sense and denotation*, ***Structure, method and meaning*** (P. Henle, H. M. Kallen, and S. K. Langer, editors), Liberal Arts Press, New York, pp. 3–24.

Alonzo Church [1951b], *The need for abstract entities*, ***American Academy of Arts and Sciences Proceedings***, vol. 80, pp. 100–113, reprinted in Martinich [1990] under the title *Intensional Semantics*.

Alonzo Church [1962], *A remark conerning Quine's paradox about modality*, Spanish version in ***Analisis Filosófico***, pp. 25–32, reprinted in English in Salmon and Soames [1988].

Alonzo Church [1973], *Outline of a revised formulation of the logic of sense and denotation, part I*, ***Nous***, vol. 7, pp. 24–33.

Alonzo Church [1974], *Outline of a revised formulation of the logic of sense and denotation, part II*, ***Nous***, vol. 8, pp. 135–156.

M. J. Cresswell [1985], ***Structured meanings: The semantics of propositional attitudes***, The MIT Press, Cambridge, Mass.

Donald Davidson [1967], *Truth and meaning*, ***Synthese***, vol. 17, pp. 304–333, reprinted in Martinich [1990] and in Davidson [1984].

Donald Davidson [1984], ***Truth and interpretation***, Clarendon Press, Oxford.

M. A. Dummett [1978], *Frege's distinction between sense and reference*, ***Truth and other enigmas***, Harvard Univ.ersity Press, Cambridge, pp. 116–144.

G. Evans [1982], ***The varieties of reference***, Clarendon Press, Oxford, Edited by J. N. McDowell.

G. Frege [1952], ***Translations from the philosophical writings of Gottlob Frege***, Blackwell, Oxford, edited by P. Geach and M. Black.

Gottlob Frege [1892], *On sense and denotation*, ***Zeitschrift für Philosophie und Philosophische Kritik***, vol. 100, Translated by Max Black Frege [1952] and also by Herbert Feigl Martinich [1990]. I have used "denotation" to render Frege's "Bedeutung," instead of Black's "meaning" or Feigl's "nominatum".

Daniel Gallin [1975], ***Intensional and higher-order modal logic***, North-Holland Mathematical Studies, no. 19, North-Holland, Elsevier, Amsterdam, Oxford, New York.

Irene Heim and Angelika Kratzer [1998], ***Semantics in generative grammar***, Blackwell.

Eleni Kalyvianaki and Yiannis N. Moschovakis [], *Two aspects of local meaning*, in preparation.

David Kaplan [1978a], *Dthat*, ***Syntax and semantics*** (Peter Cole, editor), vol. 9, Academic Press, New York, reprinted in Martinich [1990].

David Kaplan [1978b], *On the logic of demonstratives*, ***Journal of Philosophical Logic***, pp. 81–98, reprinted in Salmon and Soames [1988].

Ewan Klein and Ivan A. Sag [1985], *Type-driven translation*, ***Linguistics and Philosophy***, vol. 8, pp. 163–201.

Saul A. Kripke [1979], *A puzzle about belief*, ***Meaning and use*** (A. Margalit, editor), Reidel, pp. 239–283, reprinted in Salmon and Soames [1988].

Leonard Linsky (editor) [1971], ***Reference and modality***, Oxford University Press.

A. P. Martinich (editor) [1990], ***The philosophy of language***, second ed., Oxford University Press, New York, Oxford.

R. Montague [1970a], *English as a formal language*, ***Linguaggi nella Società e nella Tecnica*** (Milan) (Bruno Visentini et al., editors), Edizioni di Comunità, pp. 189–284, reprinted in Montague [1974].

R. Montague [1970b], *Pragmatics and intensional logic*, ***Synthèse***, vol. 22, pp. 68–94, reprinted in Montague [1974].

R. Montague [1970c], *Universal grammar*, ***Theoria***, vol. 36, pp. 373–398, reprinted in Montague [1974].

R. Montague [1973], *The Proper Treatment of Quantification in Ordinary English*, ***Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*** (J. Hintikka et al., editors), D. Reidel Publishing Co, Dordrecht, pp. 221–224, reprinted in Montague [1974].

R. MONTAGUE [1974], ***Formal philosophy***, Yale University Press, New Haven and London, Selected papers of Richard Montague, edited by Richmond H. Thomason.

YIANNIS N. MOSCHOVAKIS [1994], *Sense and denotation as algorithm and value*, ***Logic colloquium '90*** (J. Väänänen and J. Oikkonen, editors), vol. 2, Association for Symbolic Logic, Lecture Notes in Logic, pp. 210–249.

YIANNIS N. MOSCHOVAKIS [1998], *On founding the theory of algorithms*, ***Truth in mathematics*** (H. G. Dales and G. Oliveri, editors), Clarendon Press, Oxford, pp. 71–104.

JAMAL OUHALLA [1994], ***Introducing transformational grammar***, Arnold and Oxford University Press.

JUDY PELHAM AND ALASDAIR URQUHART [1994], *Russellian propositions*, ***Logic, Methodology and Philosophy of Science IX*** (D. Prawitz et al., editors), Elsevier Science.

G. PLOTKIN [1977], *LCF considered as a programming language*, ***Theoretical Computer Science***, vol. 5, pp. 223–255.

NATHAN SALMON AND SCOTT SOAMES [1988], ***Propositions and attitudes***, Oxford University Press.

D. S. SCOTT AND C. STRACHEY [1971], *Towards a mathematical semantics for computer languages*, ***Proceedings of the symposium on computers and automata*** (New York) (J. Fox, editor), Polytechnic Institute of Brooklyn Press, pp. 19–46.

J. VAN HEIJENOORT [1985], *Frege on sense identity*, ***Selected essays***, Bibliopolis, Napoli, pp. 65–70.

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF CALIFORNIA
LOS ANGELES, CA 90095-1555, USA
and
GRADUATE PROGRAM IN LOGIC, ALGORITHMS AND COMPUTATION (ΜΠΛΑ)
DEPARTMENT OF MATHEMATICS, UNIVERSITY OF ATHENS
ATHENS, GREECE
*E-mail*: ynm@math.ucla.edu