# Sense and denotation as algorithm and value

## Advanced course

**Fritz Hamm and Yiannis Moschovakis**

**ESSLLI 2010 CPH**

# A LOGIC OF MEANING AND SYNONYMY

FRITZ HAMM AND YIANNIS MOSCHOVAKIS

## Contents



These notes for a short course in ESSLLI 2010 are an update and refinement of Moschovakis [2006], which in turn was developed from a set of notes produced for a short course in NASSLLI '03. We thank Eleni Kalyvianaki for her help in preparing them.

**1. Introduction.** Frege [1879] is usually credited for introducing the basic notions of modern *mathematical* (symbolic) *logic*, which reached full maturity with the work of Hilbert, Tarski and Gödel some fifty years later. The key parts of this enterprize were the following:

(1) The choice of *First Order Language* FOL as the most suitable formal (precisely formulated) language in which to develop logic—sufficiently rich so that all mathematical theories can be expressed in it, yet simple enough so that it can be profitably studied.

(2) The precise definition of *interpretations* of FOL in *first order structures* (models) which yields a notion of *truth* for FOL-sentences, in symbols

$$\mathfrak{M} \models \theta \iff \theta \text{ is true in the structure } \mathfrak{M}.$$

(3) The precise definition of *proof* and *provability* for FOL sentences, in symbols

$$\vdash \theta \iff \text{there is a proof of } \theta$$

And finally,

(4) Gödel's proof of the *Completeness Theorem*, which identifies *logical truth* (validity) with provability,

$$\vdash \theta \iff \text{for all structures } \mathfrak{A},\ \mathfrak{A} \models \theta.$$

It is hard to overemphasize the significance for the foundations of mathematics (and science, in general) of this body of work, which answered definitively the ancient question of *what follows from what in science by logic alone*. It has also had a large number of applications in mathematics.

It was also Frege [1892] who introduced the important distinction between the *denotation* of linguistic terms (*truth value* for sentences) and their *meaning*. On

some level, this is so trivial so that it does not merit discussion: nobody would confuse

(1) "1+1 = 2" with "there are infinitely many prime numbers",

which express such obviously different, true facts about the whole numbers that it may not be worth making the difference in their meanings precise. The distinction is perhaps less immediate and more difficult to articulate for simple examples from natural language. Consider:

(2a) Abelard loved and honored Eloise,

(2b) Abelard loved Eloise and honored her,

(2c) He loved and honored Eloise,

(2d) Abelard claimed that he loved and honored Eloise,

the last two uttered in a context where "He" clearly refers to Abelard and it is commonly known that Abelard never claimed untruths. Perhaps (2a) – (2d), or at least (2a) – (2c) express the same thought, but it is difficult to make this precise—if it is true at all. Most would argue that these sentences are not quite "understood in the same way"—but making the distinctions precise is also not entirely trivial: it involves an analysis of important linguistic notions like *anaphora*, *coordination*, *logical form*, *situated interpretation* and *propositional attitudes*, and what these have to do with meaning.

The examples suggest that it may be worth developing a logical theory of *meaning and synonymy* paralleling the logical theory of truth and provability outlined in (1) – (4) above, which would account usefully for subtle differences of meaning and explain why there are none, when there are none. Plausible applications of such a theory might be found in philosophy of language and in linguistics, if not in science. Despite the extensive literature on the distinction between Frege's sense and denotation we think that the logical theory of meaning and synonymy we are advocating adds significant novel aspects to a general theory of meaning in a systematic way. Getting started on such a project was the main motivation for Moschovakis [1994], [2006]. Our aim in these lectures is to outline an extended and somewhat refined version of the main content of these two papers, with special emphasis on its relevance to linguistic analysis.[1]

The key parts of this theory corresponding to (1) – (4) for mathematical logic above are as follows, briefly and with some oversimplification:

(I) *Language.* Our formal results are about the *typed λ-calculus with acyclic recursion* $\mathsf{L}^{\lambda}_{\mathrm{ar}}$. This is an extension of the formal language used by Richard Montague in the sequence of fundamental papers Montague [1970b], [1970a], [1970c], [1973], [1974], the first, systematic and rigorous logical study of fragments of natural language. The theory we are presenting here is best viewed as an extension and refinement of Montague's work.

We emphasized above in (1) that the choice of FOL as the "most suitable formal language" was an important step in the development of mathematical

[1]For those who are familiar with Moschovakis [1994], [2006], we note that the main new contribution is the treatment of *propositional attitudes*. This requires some refinements of the theory.

logic. There is a similar need to choose a suitable formal language for the logical analysis of meaning and synonymy, and our claim here is that $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ is the best tool for this purpose.

(II) *Interpretations.* The terms of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ are interpreted in an arbitrary *higher type structure* $\mathfrak{M}$. Each term $A$ which is *closed* (with no free variables) is assigned a *denotation* (value) $\mathrm{den}(A)$, the object "named" by $A$ in $\mathfrak{M}$, very much following Montague, and we write

$$\mathfrak{M} \models A = B \iff \mathrm{den}(A) = \mathrm{den}(B) \text{ in } \mathfrak{M}. \qquad \text{(Identity)}$$

The (Fregean) novelty is that in addition to its denotation, each closed term $A$ is also assigned a *referential intension* $\mathrm{int}(A)$, an object which models the meaning of $A$ in $\mathfrak{M}$ and determines its denotation. Two terms are *synonymous* in a given structure if they have *isomorphic referential intensions*, in symbols

$$\mathfrak{M} \models A \approx B \iff \mathrm{int}(A) \cong \mathrm{int}(B) \text{ in } \mathfrak{M}. \qquad \text{(Synonymy)}$$

This is, of course, the characteristic feature of the present theory. Intuitively, $\mathrm{int}(A)$ is the natural (abstract, possibly infinitary) *algorithm*, which is determined by $A$ and *computes* $\mathrm{den}(A)$ in $\mathfrak{M}$. In slogan form:

*The meaning of a term is the algorithm which computes its denotation*

Somewhat more precisely, $\mathrm{int}(A)$ is a higher-type (set-theoretic) object which is constructed naturally from $A$ and *faithfully models* a procedure that computes $\mathrm{den}(A)$ in $\mathfrak{M}$.

(III) *The calculus of meaning and synonymy.* The classical *calculus of $\beta$-conversion* can be easily extended to $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ and provides a formal method to establish denotational identities of the form $A = B$. It is augmented here with a *reduction calculus* which establishes synonymies $A \approx B$, or, more precisely, reduces the truth of a synonymy $A \approx B$ to denotational identities between "the parts" of $A$ and $B$. It can be viewed as providing a general and rigorous specification of *the truth conditions* which ground synonymies, and it is the main technical contribution of this work—the *logical calculus of meaning and synonymy.*

(IV) *Completeness.* Combining the classical work on the $\lambda$-calculus with the methods of these notes, we can formulate and establish natural, complete axiomatizations for both (denotational) identities $A = B$ and synonymies $A \approx B$ which hold on all structures in which $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ can be interpreted. As with the logic of denotations, this will delineate those synonymies $A \approx B$ which hold *by logic alone*, and so it is of interest. On the other hand, our main interest is on understanding the relation of synonymy in *our own, interpreted language* (where love means "love" and not "hate"), and so the the completeness results for *logical synonymy* are not quite as central here as they are for mathematical logic.

Before we start with the technical development of the logic of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ in the next section, we discuss briefly some basic methodological questions which come up in projects such as these.

**1.1. Why use a formal language? Rendering.** Much of the philosophical and logical discussion of sense and denotation in the literature is formulated directly for natural language. This avoids the thorny problem of specifying the relation between natural language expressions and their "formal counterparts", or "formal expressions", or however one chooses to name the results of some generally vague *formalization process*; but it runs into the thornier problem of trying to give rigorous proofs based on the complex syntax of a specific natural language, which is not always definitively and rigorously formulated. The commonly accepted alternative, which we will adopt, is to choose a specific formal language ($\mathsf{L}^{\lambda}_{\mathrm{ar}}$) and start the logical analysis by "interpreting" in it a fragment of natural language. So every discussion of an example from natural language will start with a claim of the form

(3) every man loves some woman

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

which will be explained and motivated when not obvious, but cannot be rigorously justified, as we will not specify with any precision the all-important *rendering* (or *translation*) operation $\ldots \xrightarrow{\text{render}} A$. The corresponding *rendering claims* for the examples above are as follows, and together with (3) they exhibit all the syntactic constructs of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, which we will define precisely below:

(4a) Abelard loved and honored Eloise

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

(4b) Abelard loved Eloise and honored her

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

(4c) He loved and honored Eloise,

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

(4d) Abelard claimed that he loved and honored Eloise

$$\xrightarrow{\text{render}} \mathsf{claimed}\Big(\dot{a}, \lambda(v)[\mathsf{loved}(\dot{a},v)\ \mathsf{and}\ \mathsf{honored}(\dot{a},v)](\mathsf{Eloise})\Big)\ \mathsf{where}\ \{\dot{a} = \mathsf{Abelard}\}$$

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 further down), but also which precise reading of the expression is appropriate and what formal transformations should be made (e.g., coindexing), depending on information about "what the speaker meant", intonation, if the expression was spoken, punctuation and capitalization, if it was written, etc. We have nothing novel to say about how these factors determine the state, and so we will will assume that it is given and we will concentrate on the simpler, syntactical component of rendering

(5) natural language expression $\xrightarrow{\text{render}}$ formal expression

for which the subsequent extraction of meaning will provide some suggestions. Some would argue that what we leave out is a very important part of the extraction of meanings from linguistic expressions, and we would agree with them. On the other hand, we 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.[2] 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 reduction calculus) sets conditions and limitations on the rendering operation, it provides new ways to implement some syntactic transformations which affect meaning (like coindexing and co-ordination), and for some English phrases, it suggests some plausible, *novel renderings on* $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ which appear to correspond more accurately to our intuitive understanding of these phrases. This is an important point, and we will discuss it in some detail further down.

**1.2. Propositional attitudes.** One of the most important (and problematic) features of Frege's theory of sense is his treatment of propositional attitudes, in examples such as

(6a) George knows that $1 + 1 = 2$.

It is clear that the truth of this depends not only on the truth value of "$1+1 = 2$" but also on its sense; otherwise (6a) would be equivalent with

(6b) George knows that there are infinitely many prime numbers,

which need not be true if George knows a bit of rudimentary arithmetic but is ignorant of Euclid's theorem about the infinitude of primes. To account for this without abandoning his general *compositionality principle for denotations*, Frege postulates that in such *indirect occurrences*, the denotation of a sentence which is the object of a belief assertion is its (customary) sense. The doctrine leads to the introduction of many layers of "indirect senses" and (at the minimum) complicates greatly the development of a coherent, rigorous logical theory of meaning.[3]

Montague avoids these complications by adopting a modal interpretation of propositional attitudes. Briefly, every sentence $\theta$ is assumed to have a truth value in every "possible world", and then George knows $\theta$ if $\theta$ is true in all worlds which are "accessible" to him. There is a well-known problem with this account: one

[2]Cf. the discussion and the references to Dummett [1978] and Evans [1982] in the introduction to Moschovakis [1994]. See also Tichý [1969], which was not known to Moschovakis when he started on this project.

[3]It also conflicts with the most plausible motivation for introducing a compositionality principle for denotations, which is to develop a logic of denotations independent of senses: this plainly fails if among the denotations that we need to introduce are precisely the senses we were trying to avoid—and many more "indirect versions" of them.

would assume that "$1 + 1 = 2$" and "there are infinitely many primes" are both true in all (plausible) possible worlds, and so George could not coherently know one and not the other. Matters are worse for simpler propositional attitudes: it is not easy to account modally for the different truth values of the two sentences

(7a) George claimed that he didn't kill the judge

(7b) George claimed that he was in Baltimore at the time

if, indeed, the claimed facts are both true—but George did not claim the second one, perhaps because he didn't want his wife to know that he had been to Baltimore.

Referential intension theory makes possible a treatment of propositional attitudes which is ultimately very close to Frege's but avoids explicitly introducing indirect senses. This is because referential intensions are higher type objects to which $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ can refer: and so we can compute the truth values and the referential intensions of (7a) and (7b) from

int(he didn't kill the judge) and int(he was in Baltimore at the time)

and render (7a) and (7b) in $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ in full agreement with our intuitions about claiming. This, however, involves some technicalities which will be easier to explain after we understand the denotational part of the theory, so we will put it off.

The plan then is to develop first the syntax and semantics of *the denotational part of* $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ which does not have *attitudinal constants* like know that, claim that, and then *interpret* (translate) the full language into its denotational part.

**1.3. Is all language situated? Local meaning and synonymy.** To understand and assign a truth value to

(8) She loves me

we must know (at least) who the speaker is, who "she" is, and when the sentence was uttered. We have already mentioned that in formal studies of semantics, this information is assumed to be extracted (somehow) from the informal context and bundled into a state $a$, which then together with the formal rendering of (8) determines at least the truth value and perhaps also its meaning as an *utterance*.

Montague includes the "possible world" in the state and goes one step further: he assumes that we can only understand language *in context* and, in particular, that it makes no sense to refer to "the denotation of $A$" without specifying the state in which $A$ is evaluated. Formally, every closed term $A$ of Montague's *language of intensional logic* LIL which expresses an assertoric sentence of natural language is interpreted by its *Carnap intension*

(9) $\mathrm{CI}(A) : \mathrm{States} \to \mathrm{Truth\ values}$,

which assigns a truth value $\mathrm{CI}(A)(a)$ to $A$ in every state. The slogan is

*all language is situated*

and it has been generally assumed in most studies of semantics since the 1970s.[4]

But is the slogan true? Consider again a typical mathematical claim,

(10a) $P \equiv$ there are infinitely many prime numbers,

[4]Barwise was a frequent and eloquent exponent of this view.

which can certainly be uttered in English and so its meaning and truth value should be considered along with those of claims about George and "him". Most would agree that the truth value of $P$ is independent of the state, this being, after all, an important attribute of mathematical claims. So one expects (and one gets) in LIL

(10b) for every state $a$, $\mathrm{CI}(P)(a) = \mathrm{true}$

in Montague semantics, and $P$ is called *rigid*, meaning that its Carnap intension is constant. Rigidity is an important notion which has been extensively investigated in less trivial circumstances that do not concern us here. (We will consider it further down.)

The question is whether (10b) is equivalent with the absolute claim

(10c) $P$ is true

which assumes, in effect, that $P$ is interpreted without reference to context. One can appeal to standard philosophical arguments that may justify different readings of (10b) and (10c)—that the first views $P$ as contigently (if rigidly) true while the second takes it to be analytic, etc.—and, as always, there are objections and objections to the objections.

For our purposes, what matters is not so much the equivalence of these two claims, but *the possible difference in the logical meaning of* $P$ when we understand it as having a single truth value or as determining a constant function on the set of states—especially *the contribution that* $P$ *makes to the meaning of sentences in which it occurs*; and as it turns out, this contribution changes, in subtle but important ways. We will consider this in some detail further down. For now, let us just say that with each term $A$ like (8) which naturally defines a function from states to truth values and with each state $a$, we will associate a *local referential intension* $\mathrm{int}(A)(a)$ and a *local denotation* $\mathrm{den}(A)(a)$ of $A$ at the state $a$, in symbols

$$\mathfrak{M} \models A =_a B \iff \mathrm{den}(A)(a) = \mathrm{den}(B)(a) \text{ in } \mathfrak{M}, \qquad \text{(Local identity)}$$

$$\mathfrak{M} \models A \approx_a B \iff \mathrm{int}(A)(a) \cong \mathrm{int}(B)(a) \text{ in } \mathfrak{M}. \qquad \text{(Local synonymy)}$$

**2. Formal syntax of $\mathsf{L}^{\lambda}_{\mathbf{ar}}$.** The language $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ is a *typed calculus of terms*, an extension of the *two-sorted type theory* $\mathsf{Ty}_2$ of Gallin [1975][§8] into which the language of *intensional logic* LIL of Montague [1973] can be interpreted by Gallin's Theorem 8.2.[5,6] Its syntax is determined by the *types*, the *constants*, the *variables* and the *terms*.

[5]Gallin showed that each term of LIL is denotationally equal (in a suitable sense) with a term of $\mathsf{Ty}_2$, and Zimmermann [1989] showed that, conversely, every relevant term of $\mathsf{Ty}_2$ is denotationally equal to a term of LIL. Partly because of these results, most students of Montague's semantics have adopted the easier to deal with $\mathsf{Ty}_2$ as their formal language of choice. Kalyvianaki and Moschovakis [2008] combine Gallin's interpretation with the methods of these lectures to assign natural, robust meanings to the terms of LIL.

[6]Montague's 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 $\sigma$ denotes its *Carnap intension*, a function $\mathrm{CI}(A) : \mathbb{T}_{\mathsf{s}} \to \mathbb{T}_{\sigma}$ from the states to the objects of type $\sigma$. In effect, "all language is local (situated)" in his approach, it is simply not allowed to consider sentences independently of some, specific context. Perhaps Montague took this route because we cannot explicitly refer to the state in natural language; but, like every formal language, LIL has many terms which do not render anything humans would utter, including free variables. The lack of state variables causes considerable

**2.1. Types.** These 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,[7]

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

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:

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

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

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

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

It is also useful to introduce the notation

(13) $\tilde{\mathsf{q}} :\equiv ((\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{t}}) \equiv$ 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 mathematical 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 distinct.

**2.2. Constants.** We assume given a set $K$ of typed denotational constants, the *denotational lexicon*, and we write

$$\mathsf{c} : \sigma \iff \mathsf{c} \text{ is of type } \sigma,$$

so that

$$17 : \mathsf{e}, \quad \mathsf{John} : \tilde{\mathsf{e}}, \quad \mathsf{man} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}.$$

Some examples of denotational empirical constants are given in Table 1. Table 2 lists the usual logical constants in two versions, one with pure types to be used in rendering state-independent (especially mathematical) sentences and another for combining terms which denote Carnap intensions and individual concepts. The distinction will made clear when we define their denotations further down.

awkwardness in the development of logic, and the absence of pure (state-free) types interferes with the development of a satisfactory theory of meaning for mathematical statements.

[7] We 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, a constant denoting the identity relation between the objects that $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ is about.

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

TABLE 1. Denotational empirical constants.

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 $K$.

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

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

Intuitively, pure variables are used *to define functions* and they are bound by the $\lambda$-operator, the characteristic construct of the $\lambda$-calculus. Locations are *assigned values* and they are bound by the $\mathsf{where}$ operation, the characteristic construct of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$. The use of these two, parallel sets of variables which play different roles is essential for the correct definition of referential intensions.

**2.4. Terms.** These are defined recursively, starting with the variables and the constants and using *application*, $\lambda$-*abstraction* and (mutual) *acyclic recursion* (the $\mathsf{where}$ construct). The definition also assigns a type to every term and specifies the free and bound occurrences of variables in it. We write

$$A : \sigma \iff A \text{ is a term of type } \sigma.$$

Table 3 summarizes the definition, which we now explain in detail.

**2.4.1. Constants and variables.** *Each denotational constant* $\mathsf{c}$ *and each variable* $x$ *of either kind is a term of the type of* $x$*; the term* $\mathsf{c}$ *has no free variables, and* $x$ *is the only free occurrence of a variable in the term* $x$*.*

For example:

$$17 \xrightarrow{\text{render}} 17 : \mathsf{e}, \quad \text{John} \xrightarrow{\text{render}} \mathsf{John} : \tilde{\mathsf{e}}, \quad \text{man} \xrightarrow{\text{render}} \mathsf{man} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}.$$

**2.4.2. Application:** *If* $A : \sigma \to \tau$ *and* $B : \sigma$*, then* $A(B) : \tau$*; a variable occurs free in* $A(B)$ *if it occurs free in either* $A$ *or* $B$*.*

For example:

$$\begin{aligned}
\text{John is a man} &\xrightarrow{\text{render}} \mathsf{man}(\mathsf{John}) : \tilde{\mathsf{t}} \\
\text{tall man} &\xrightarrow{\text{render}} \mathsf{tall}(\mathsf{man}) : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}} \\
\text{John is a tall man} &\xrightarrow{\text{render}} \mathsf{tall}(\mathsf{man})(\mathsf{John}) : \tilde{\mathsf{t}} \\
\text{Abelard loved} &\xrightarrow{\text{render}} \mathsf{loved}(\mathsf{Abelard}) : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}} \\
\text{Abelard loved Eloise} &\xrightarrow{\text{render}} \mathsf{loved}(\mathsf{Abelard})(\mathsf{Eloise}) : \tilde{\mathsf{t}}
\end{aligned}$$

| | | | | | |
|---|---|---|---|---|---|
| $=_\sigma$ | : | $\sigma \times \sigma \to \sigma$ | $\mathsf{not}, \Box, \mathsf{in\ the\ future}$ | : | $\tilde{\mathsf{t}} \to \tilde{\mathsf{t}}$ |
| $\neg$ | : | $\mathsf{t} \to \mathsf{t}$ | $\mathsf{and}, \mathsf{or}, \mathsf{if}\ ..\ \mathsf{then}\ ..$ | : | $\tilde{\mathsf{t}} \times \tilde{\mathsf{t}} \to \tilde{\mathsf{t}}$ |
| $\&, \vee, \Rightarrow$ | : | $\mathsf{t} \times \mathsf{t} \to \mathsf{t}$ | $\mathsf{every}, \mathsf{some}$ | : | $(\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{q}}$ |
| $\forall_\sigma, \exists_\sigma$ | : | $(\sigma \to \mathsf{t}) \to \mathsf{t}$ | $\mathsf{the}$ | : | $(\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{e}}$ |

TABLE 2. Logical constants.

In accordance with the currying convention discussed above, if

$$A : \sigma_1 \times \sigma_2 \to \tau,\ B : \sigma_1, \quad \text{and } C : \sigma_2,$$

we write synonymously

(14) $A(B, C) \equiv A(B)(C), \quad A(B, C, D) \equiv A(B)(C)(D), \quad \ldots$

so that the last example above would be written

(15) Abelard loved Eloise $\xrightarrow{\text{render}}$ $\mathsf{loved}(\mathsf{Abelard}, \mathsf{Eloise}) : \tilde{\mathsf{t}}$

which is a little easier to read.

It is best to understand (14), (15) and (17) below as "misspellings" of formal terms, or informal instructions for constructing a formal, syntactically correct term rather than try to incorporate them into the precise specification of the syntax. The practice is very common in logic, and it extends to the use of abbreviations, omitting parentheses or replacing then by brackets and other delimiters (of various sizes), using infix rather than the official "function first arguments next" notation, and many other, similar devices. For example:

$[A(B) = C]\ \mathsf{and}\ [D(C)] \equiv \mathsf{and}(= (A(B))(C))(D(C))$

John's sister is Mary and Mary is tall

$$\xrightarrow{\text{render}} \mathsf{sister}(\mathsf{John}) = \mathsf{Mary}\ \mathsf{and}\ \mathsf{tall}(\mathsf{Mary}) \equiv \mathsf{and}(= (\mathsf{sister}(\mathsf{John}))(\mathsf{Mary}))(\mathsf{tall}(\mathsf{Mary})).$$

It will also be useful to allow on occasion the *dummy recursion construct* $A\ \mathsf{where}\ \{\ \}$ as a misspelling of $A$, i.e.,

(16) $A\ \mathsf{where}\ \{\ \} \equiv_{\text{def}} A.$

**2.4.3. λ-abstraction:** *If $B : \tau$ and $v$ is a pure variable of type $\sigma$, then*

$$\lambda(v)(B) : \sigma \to \tau;$$

*a variable $x$ occurs free in $\lambda(v)(B)$ if it occurs free in $B$ and it is not $v$.*

Intuitively, $\lambda(v)(B)$ denotes a function $f$ such that for each object $\overline{v}$ (of the appropriate type), $f(\overline{v})$ is the value of $B$ when $v$ refers to $\overline{v}$. To deal effectively with functions of more than one variables following the "currying" technique, we write

(17) $\lambda(u, v)(A) :\equiv \lambda(u)(\lambda(v)(A)) : \tilde{\sigma}_1 \times \tilde{\sigma}_2 \to \tau$

and similarly with more variables.

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

$\mathsf{c}$ 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 $\lambda(v)(B) : (\sigma \to \tau)$

$n \geq 1$, $A_i : \sigma_i$, $\dot{p}_1, \dots, \dot{p}_n$ are distinct recursion variables , $\dot{p}_i : \sigma_i$, the system $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$ is acyclic, and $A_0$ where $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\} : \sigma_0$

In addition (recursively) all occurrences of $v$ are bound in $\lambda(v)(B)$, and all occurrences of $\dot{p}_1, \dots, \dot{p}_n$ are bound in $A_0$ where $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$; occurrences of variables not bound by this clause are free.

TABLE 3. The denotational terms of $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$.

Pure variables and the $\lambda$-abstraction construct are used to express *coordination*. For example:

$$\text{tall and handsome} \xrightarrow{\text{render}} \lambda(x)(\mathsf{tall}(x) \text{ and } \mathsf{handsome}(x)) : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$$

$$\text{John is tall and handsome} \xrightarrow{\text{render}} \lambda(x)(\mathsf{tall}(x) \text{ and } \mathsf{handsome}(x))(\mathsf{John}) : \tilde{\mathsf{t}}$$

$$\text{loved and honored} \xrightarrow{\text{render}} \lambda(u,v)\Big(\mathsf{loved}(u,v) \text{ and } \mathsf{honored}(u,v)\Big) : \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$$

The last of these is used to construct the rendering (4a),

Abelard loved and honored Eloise

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

**2.4.4. Acyclic recursion:** *If* $A_0 : \sigma_0, A_1 : \sigma_1, \dots, A_n : \sigma_n$ *and*

$\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$

*is an acyclic system of assignments* (as explained below), *then*

$A_0$ where $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\} : \sigma_0$.

*A variable $x$ is free in this term if it is free in some $A_i$ and it is not one of the locations $\dot{p}_1, \dots, \dot{p}_n$.*

The best, intuitive way to understand the recursive construct is to read "where" more-or-less normally:

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

communicates the same information as

$j$ loves $\dot{s}$, where $j$ is John, $\dot{s}$ is the sister of $\dot{m}$ and $\dot{m}$ is Mary

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

in other words "John loves Mary's sister"—and, as we will see later, 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* $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$ is *acyclic* if it is possible to associate a natural number $\mathrm{rank}(\dot{p}_i)$ with each of the locations, so that

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

the obvious idea is that $\dot{p}_i$ has higher rank than $\dot{p}_j$ if its value "depends" (or could depend) on that of $\dot{p}_j$. For example, the system

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

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

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

because any ranking of $\dot{p}$ would need to satisfy $\mathrm{rank}(\dot{p}) > \mathrm{rank}(\dot{p})$, which it cannot. Acyclic systems express "trivial" systems of "recursive definitions" which "close off" (and produce a value) in a finite number of steps.[8]

Acyclic recursion can be used to render faithfully anaphora. For example, consider a preliminary, literal formalization

Abelard loved Eloise and (he) honored her

$$\xrightarrow{\text{formalize}} \mathsf{loved}(\mathsf{Abelard}, \mathsf{Eloise}) \text{ and } \mathsf{honored}(\mathsf{he}, \mathsf{her}).$$

This could, in fact, be what we want if "he" were not omitted and referred to Abelard's friend George and "her" referred to George's wife. It is not the intended reading, of course, and we would like to *coindex* "he" with "Abelard" and "her" with "Eloise". Doing these one step at a time by the recursion construct, we get

(18) $\mathsf{loved}(\mathsf{Abelard}, \mathsf{Eloise})$ and $\mathsf{honored}(\mathsf{he}, \mathsf{her})$

$\xrightarrow{\text{coindex}}$ $\mathsf{loved}(\dot{a}, \mathsf{Eloise})$ and $\mathsf{honored}(\dot{a}, \mathsf{her})$ where $\{\dot{a} = \mathsf{Abelard}\}$

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

The reduction calculus will justify one additional step in this sequence of transformations which combines the two occurrences of where and leads to the

[8] If we remove the acyclicity restriction on the recursion construct, we obtain the *λ-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 these lectures 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 the language FLR introduced in Moschovakis [1989] 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.

rendering in (4b)

(19) Abelard loved Eloise and honored her

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

Whatever the formal justification of this computation (which we will give later), it seems that it produces a faithful representation of the English sentence. Notice, for example, that the *head*

$$\mathsf{loved}(\dot{a},\dot{e}) \text{ and } \mathsf{honored}(\dot{a},\dot{e})$$

of the formal term is a *conjunction*, much as the English sentence

(20) Abelard loved Eloise and honored her

is a conjunction; so this rendering *preserves the logical form* of the English sentence. It is not clear that this sentence can be rendered faithfully without the use of the recursion construct, by using the $\lambda$ to capture the anaphora: the closest we can get (as far as we can tell) is

(21) $\lambda(u,v)\Big(\mathsf{loved}(u,v) \text{ and } \mathsf{honored}(u,v)\Big)(\mathsf{Abelard},\mathsf{Eloise})$.

This is a predication rather than a conjunction, and it gives the same renderings for (20) and

(22) Abelard loved and honored Eloise,

which does not seem right. (Cf. (4a) and (4b).)

*One of our main aims is to give a useful, rigorous account of the logical form of sentences, which*, we assume, *is an important part of logical meaning.*

This completes the definition of terms.

**2.5. Explicit, recursive and $\lambda$-calculus terms.** A term $A$ is **explicit** if the recursion construct where does not occur in it; it is **recursive** if it is of the form $A_0$ where $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$; and it is 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 $\mathsf{Ty}_2$.

A term is **closed** if it has no free occurrences of variables.

**2.6. Term 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 $\pi : \{1,\dots,n\} \to \{1,\dots,n\}$,

$$A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\} \equiv_c A_0 \text{ where } \{\dot{p}_{\pi(1)} := A_{\pi(1)}, \dots, \dot{p}_{\pi(n)} := A_{\pi(n)}\},$$

so that, for example,

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

The last condition means that the assignments within { } are interpreted as a *set*, not a sequence.

**Fact.** *If $A \equiv_c B$, then the same constants occur in $A$ as they occur in $B$, and the same variables occur free in $A$ as they occur in free in $B$.*

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.

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

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

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

(S1) *Each $\mathbb{T}_\sigma$ is a set.* We further assume that there is at least one state and at least three truth values, 0 (falsity), 1 (truth) and *er* (error), and that (for convenience) that truth values are entities,

$$0, 1, er \in \mathbb{T}_{\mathsf{t}} \subseteq \mathbb{T}_{\mathsf{e}}.$$

We will discuss the role of the *er* truth value in Section 3.3 below.

(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 $c \in \mathbb{T}_\sigma$.*

(S4) den *is a function which associates with each term $A : \sigma$ and each valuation* g *of the variables an object* $\mathrm{den}(A)(\mathrm{g}) \in \mathbb{T}_\sigma$ *so that the following conditions hold*:[9]

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

We will explain this last unfamiliar clause with an example a bit further down.

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 $A$. Using familiar notation from logic, we write

(23a) $\mathfrak{M}, \mathrm{g} \models A = B \iff \mathrm{den}(A)(\mathrm{g}) = \mathrm{den}(B)(\mathrm{g})$ in $\mathfrak{M}$,

(23b) $\mathfrak{M} \models A = B \iff$ for all valuations $\mathrm{g}, \mathfrak{M}, \mathrm{g} \models A = B$

[9] A *valuation* is a function g which assigns to each pure or recursion variable $x$ of type $\sigma$ an object $\mathrm{g}(x) \in \mathbb{T}_\sigma$. (These are often called *assignments* in logic, but we will use "valuation" to avoid confusion with the formal, syntactic assignments in the acyclic recursion construct.) If $x : \sigma$ is a variable and $t \in \mathbb{T}_\sigma$, then the *update* of g by *revision* $x := t$ is defined in the obvious way,

$$\mathrm{g}\{x := t\}(y) = \begin{cases} t, & \text{if } y \equiv x, \\ \mathrm{g}(y), & \text{otherwise.} \end{cases}$$

The structure $\mathfrak{M}$ 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 structure $\mathfrak{M}$ 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.

**3.1. The denotation of recursive terms.** To illustrate the computation of denotations in the recursive case, consider the closed term

(24) $A \equiv \dot{p}$ and $\dot{q}$ where $\{\dot{p} := \mathsf{loves}(j, \dot{m}), \dot{q} := \mathsf{dislikes}(j, \dot{h}),$
$\dot{h} := \mathsf{husband}(\dot{m}), j := \mathsf{John}, \dot{m} := \mathsf{Mary}\}.$

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

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

At this point we are tempted to infer that

(25a) den$(A)$ = the truth value of
"John loves Mary and he dislikes her husband",

and this is almost true—but not quite. This is because $A : \tilde{\mathsf{t}}$, and so den$(A)$ is not a truth value but a function on states to truth values, den$(A) : \mathbb{T}_{\mathsf{s}} \to \mathbb{T}_{\mathsf{t}}$. The computation (with the side remarks omitted or made state-dependent) is correct, because of the following typings of the constants and their interpretations:

$\mathsf{John} : \tilde{\mathsf{e}}$, John$(a)$ = the object identified as John in state $a$,
$\mathsf{Mary} : \tilde{\mathsf{e}}$, Mary$(a)$ = the object identified as Mary in state $a$,
$\mathsf{husband} : \tilde{\mathsf{e}} \to \tilde{\mathsf{e}}$, husband$(x)(a)$ = the husband of $x(a)$ in state $a$,
$\mathsf{loves} : \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$, loves$(x, y)(a) \iff x(a)$ loves $y(a)$ in state $a$,
$\mathsf{dislikes} : \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$, dislikes$(x, y)(a) \iff x(a)$ dislikes $y(a)$ in state $a$
$\mathsf{and} : \tilde{\mathsf{t}} \times \tilde{\mathsf{t}} \to \tilde{\mathsf{t}}$, and$(x, y)(a) \iff x(a)$ and $y(a)$ are both true in state $a$.

So in the end we get that

(25b) den$(A)(a)$ = the truth value of
"John loves Mary and he dislikes her husband" in state $a$,

which is the correct version of (25a).

**3.2. Our universe.** To interpret natural language in $\mathfrak{M}$, 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 possible world), trees, points in spacetime, etc.[10]

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 4, 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 $\mathbb{T}_{\mathsf{s}}$.

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 $\mathfrak{M}$ 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 readings of natural language phrases.*

*We now fix one* (possibly non-standard) *structure*

$$\mathfrak{M}_0 = \text{the intended interpretation,} \qquad \text{(our universe)}$$

without placing on it any restrictions beyond (S1) – (S4). Our discussions of various possibilities on the nature of states, the existence and nature of entities, etc., will refer to this one, fixed structure $\mathfrak{M}_0$.

The members of $\mathbb{T}_\sigma$ in our universe are the *objects of type* $\sigma$, and we will write synonymously

$$x : \sigma \iff x \in \mathbb{T}_\sigma.$$

**3.3. Errors and presuppositions.** The inclusion $\mathbb{T}_{\mathsf{t}} \subseteq \mathbb{T}_{\mathsf{e}}$ 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 "John dislikes Mary's husband" in a state $a$, if Mary is not married or has more than one husband in $a$.

It takes a little care to interpret the constants so that this method works and gives the desired denotations, but the idea is simple and we define here only empirical conjunction and existential quantification,

$$\mathrm{and} : \tilde{\mathsf{t}} \times \tilde{\mathsf{t}} \to \tilde{\mathsf{t}}, \qquad \mathrm{some} : (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \times (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}) \to \tilde{\mathsf{t}}$$

[10] 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.

as examples:

$$\mathrm{and}(x,y)(a) = \begin{cases} 1, & \text{if } x(a) = y(a) = 1, \\ 0, & \text{if either } x(a) = 0 \text{ or } y(a) = 0, \\ er, & \text{otherwise,} \end{cases}$$

$$\mathrm{some}(x)(y)(a) = \begin{cases} 1, & \text{if there is some } y \in \mathbb{T}_{\tilde{\mathsf{e}}} \text{ such that } x(y)(a) = 1, \\ 0, & \text{if for every } y \in \mathbb{T}_{\tilde{\mathsf{e}}},\ x(y)(a) = 0, \\ er, & \text{otherwise.} \end{cases}$$

This makes "Mary's husband is tall and $1 + 1 = 0$" false (rather than error) in a state where Mary does not have a husband, and it makes "some horse is a senator" true in Nero's time, even though for most other states it would be assigned *er*, the property of being a senator not normally applying to horses.

It appears that this simple device handles correctly most simple cases of what Soames [1989] calls (Fregean) *logical presupposition*, at least in the sense of providing the expected truth values. We will illustrate the technique with additional examples further down, but we will not go into a detailed analysis of the complex issue of presupposition.

**3.4. Formal replacement and denotational compositionality.** 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 valuation 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

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

From this we get very easily (by induction on $A$) the following basic

**Theorem** (Denotational Compositionality)**.** *Suppose $A$ is a term, $x$ is a variable or constant of type $\sigma$, and $C_1, C_2$ are terms of type $\sigma$ which are free for $x$ in $A$. Then for all structures $\mathfrak{M}$ and all valuations* g*,*

$$\text{if } \mathfrak{M}, \mathrm{g} \models C_1 = C_2, \text{ then } \mathfrak{M}, \mathrm{g} \models A\{x :\equiv C_1\} = A\{x :\equiv C_1\}.$$

We will tacitly assume that all replacements are free, when we appeal to this result.

**3.5. $\mathsf{L}^{\lambda}_{\mathbf{ar}}$ vs. the typed $\lambda$-calculus $\mathbf{Ty}_2$.** 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* $\mathsf{Ty}_2$, for example the term $A$ in (4b) is not referentially synonymous with any explicit term (so long as the constants which occur in it are not given truly perverse interpretations).

**4. Examples.** Beyond fleshing out some of the formal definitions of Section 2, the simple examples in this section will help illustrate the modeling of meaning coming up next. We will use the notations $A \approx B$ for synonymy and $A \approx_a B$ for synonymy in state $a$, in our universe, which we introduced in Section 1 and which we will define formally in Section 8.

First a cautionary note.

**4.1. $\beta$-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 for every structure $\mathfrak{M}$,*

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

For example,

$$\mathfrak{M} \models \Big(\lambda(\jmath)\mathsf{loves}(\jmath,\jmath)\Big)(\mathsf{John}) = \mathsf{loves}(\mathsf{John},\mathsf{John}).$$

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

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

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

(26a) John loves himself $\xrightarrow{\text{render}}$ $\Big(\lambda(\jmath)\mathsf{loves}(\jmath,\jmath)\Big)(\mathsf{John})$,

(26b) John loves John $\xrightarrow{\text{render}}$ $\mathsf{loves}(\mathsf{John},\mathsf{John})$.

One can argue for this difference in meaning by appealing to the Compositionality Principle, that to understand (26a) you need to know the word himself which does not occur in (26b). From the computational point of view, to compute the truth value of (26b) you need to compute the reference of John twice, while to decide (26a) you need to compute that reference only once.[11]

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 appeal to it unthinkingly at the rendering stage, building false formal synonymies before we get started with the analysis in $\mathsf{L}^{\lambda}_{\mathrm{ar}}$.

**4.2. States.** To be specific, we will assume in these notes (basically following Montague) that a *state* is a quadruple

$$a = (i, j, k, A, \delta)$$

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

[11] If "computing John" seems trivial, consider instead the example from arithmetic

$$\lambda(x)(x + x)(\pi) = \pi + \pi.$$

The term on the right asks for computing twice the infinite decimal $\pi$ and then adding the two values, while the term on the left requires computing $\pi$ only once, and then applying the "doubling function" to it. These two terms express different algorithms for computing the number $2\pi$.

occurrences of proper names and indexicals, 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 understand and determine the truth value of

"John loves her and she loves him",

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 $a$ by

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

and *we will leave open the question of which states exist* in the basic set $\mathbb{T}_{\mathsf{s}}$ of our universe $\mathfrak{M}_0$, in accordance with the discussion in 3.2. The choice of $\mathbb{T}_{\mathsf{s}}$ does not affect the way in which the denotations and referential intensions of terms in $\mathfrak{M}_0$ 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 sometimes touch on difficult questions of philosophy and the expressibility of natural language.

**4.3. Pure and natural language types and terms.** In accordance with the discussion in Section 1.3, terms which denote mathematical objects are assigned pure types, and more specifically, mathematical statements are rendered by terms of type $\mathsf{t}$. For example, assuming that the relevant constants are in the lexicon,

there is a set with no members

$$\xrightarrow{\text{render}} (\exists x)[\mathsf{Set}(x)\,\&\,(\forall y)\neg(y \in x)]$$
$$\equiv \exists_{\mathsf{e}}\Big[\lambda(x)\Big[\mathsf{Set}(x)\,\&\,\forall_{\mathsf{e}}\Big(\lambda(y)\neg(\in\,(y,x))\Big)\Big]\Big] : \mathsf{t}.$$

Mathematicians—especially logicians—are very familiar with this simple rendering operation, and they do it (or they assume that it can be done) without thinking; sometimes they are even accused that of "thinking in formal predicate logic". So we have little to say about it.

On the other hand, $\mathsf{He} : \tilde{\mathsf{e}}$, because the "He" of one sentence may be different from the "He" of another, and so $\mathsf{He}$ denotes a function on states, an individual concept. Even $\mathsf{John}$ may refer to different persons in different states, which is why we set $\mathsf{John} : \tilde{\mathsf{e}}$. In fact, all natural renderings of English phrases have *natural language*—or just *natural types*, defined as follows:

$$\sigma :\equiv \tilde{\mathsf{e}} \mid \tilde{\mathsf{t}} \mid (\sigma_1 \to \sigma_2) \qquad \text{(natural language types)}$$

All the empirical constants in Table 1 and the natural, logical constants in Table 2 have natural types, and so (easily) all the terms which are constructed from

them *using variables of natural types* have natural types. These are the *natural language terms*, those which all English phrases; and in agreement with the *all language is situated* principle, they denote functions from the states to objects.

Well, almost all: how about the following, which many of us who have taught remedial algebra have been tempted to utter:

If $\sqrt{a+b} = \sqrt{a} + \sqrt{b}$, then I will hang from the chandellier.

We want to render this by $\mathsf{if}\ A\ \mathsf{then}\ B$ with suitable $A : \mathsf{t}$ and $B : \tilde{\mathsf{t}}$, which does not make sense according to the typing of the natural implication; so we *raise the type* of the hypothesis in the obvious way and set

If $\sqrt{a+b} = \sqrt{a} + \sqrt{b}$, then I will hang from the chandelier

$$\xrightarrow{\text{render}}\ \mathsf{if}\ \lambda(u)A\ \mathsf{then}\ B$$

where $u : s$. (In practice, of course, we won't even bother to show explicitly this simple type-raising.)

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

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

where $b \mapsto y$ is the constant function on the states with value $y$. 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,[12]

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

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 $\mathrm{tall}(er) = er$—as it should be. And "the King of France is bald" will also be assigned *er* today, contrary to Russell's wishes.[13]

**4.5. Carnap objects of type $(\mathsf{s} \to \sigma)$; rigidity.** These include the denotations of demonstratives $\mathsf{He}, \mathsf{her}, \mathsf{today}, \ldots$, of type $\tilde{\mathsf{e}}$, as well as the *Carnap intensions* of type $\tilde{\mathsf{t}}$, for example "it rains", for which[14]

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

[12]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 $\mathsf{tall}$, and for this we must depend on the informal context before rendering; it is assumed here that Mary's husband is classified as tall among men and not (for example) among basketball players.

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

[14]In Montague's $\mathsf{LIL}$, every term denotes a Carnap object. In $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, the only Carnap objects of natural type are the individual concepts and the Carnap intensions, of respective types $\tilde{\mathsf{e}}$ and $\tilde{\mathsf{t}}$. (This is immediate from the definition of natural types.)

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[15]

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

then the object $\mathrm{dere}_\sigma(x, a) : (\mathsf{s} \to \sigma)$ 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.

**4.6. 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:

$$\Box(p)(a) \iff (\forall b)p(b) \quad (p : \tilde{\mathsf{t}}).$$

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 understand his interpretation as a de re reading of the modality, without forbidding the de dicto reading.

For the de re interpretation

$$\Box_1(p, x)(a) \iff x(a) \text{ necessarily has property } p \quad (p : \tilde{\mathsf{t}}, x : \tilde{\mathsf{e}}),$$

the correct definition of $\Box_1$ is

$$\Box_1(p, x)(a) = \Box(p(\mathrm{dere}(x, a)))(a) \quad (p : \tilde{\mathsf{t}}, x : \tilde{\mathsf{e}}).$$

This gives the two possible renderings

(28a) The President is necessarily American

$$\xrightarrow{\text{render}} \Box(\mathsf{American}(\mathsf{the}(\mathrm{President}))$$

(28b) The President is necessarily American

$$\xrightarrow{\text{render}} \Box_1(\mathsf{American}, \mathsf{the}(\mathsf{President}))$$

which express the following two natural readings of the English phrase uttered in February 2009:

(29a) The President (of the US) is necessarily an American

(29b) Obama is necessarily American.

For binary modal operators, the de re interpretation in both variables is

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

[15] This is really Kaplan's $\mathrm{dthat}(x, a)$ in Kaplan [1978a], but we are using a different notation to avoid confusion because Kaplan's understanding (and application) of this important function is somewhat different from the present one. The notation $\mathrm{dere}(x, a)$ comes from the use we will make of these functions further down to derive the de re readings of modal operators from their simpler de dicto versions. We are not assuming, however, that language has a constant $\mathsf{dere}_\sigma$, for any $\sigma$: we would not know the English word for it, and $\mathrm{dere}_\sigma : (s \to \sigma) \times \mathsf{s} \to (\mathsf{s} \to \sigma)$, which is not a natural language type.

and it can be defined using $\Box$ and the dere function by

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

For example, with primitives $\mathsf{reside} : \tilde{\mathsf{e}} \times \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ for "$x$ is in place $y$" and $\mathsf{here} : \tilde{\mathsf{e}}$ a constant such that $\text{here}(a) = \text{location}(a)$, this allows four renderings of

I am necessarily here

as follows:

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

Uttered by Obama in Washington in February 2009, the first of these conveys the information that "the speaker is necessarily at the place the utterance is made", and the last one that "Obama necessarily resides in Washington". 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 for the de dicto and the de re readings of in the past, in all possible worlds (but at the present time and location), etc.

**4.7. Local and modal dependence.** The value of $\Box_1(p, x)(a)$ depends on all the values of its first argument $p : (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}})$ but only on the *local* value $x(a)$ of its second argument. In general, for $n$-ary functions, an object

$$p : (\mathsf{s} \to \sigma_1) \times \cdots \times (\mathsf{s} \to \sigma_n) \to (\mathsf{s} \to \tau)$$

is *local on its i'th argument* if for each state $a$ and all $x_1, \ldots, x_n$, the value $p(x_1, \ldots, x_n)(a)$ depends only on $x_i(a)$. For the unary case, this means that $p$ is local if

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

and for the binary case,

$$p \text{ is local in its first argument} \iff p(x, y)(a) = p(\text{dere}(x, a), y)(a),$$
$$p \text{ is local in its second argument} \iff p(x, y)(a) = p(x, \text{dere}(y, a))(a).$$

An object is *modal in its i'th argument* if it is not local in that argument.[16]

A closed term $A : (\mathsf{s} \to \sigma_1) \times \cdots \times (\mathsf{s} \to \sigma_n) \to (\mathsf{s} \to \tau)$ is local or modal in an argument accordingly as $\text{den}(A)$ has the corresponding property.

[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 our approach.

For example, the natural logical operations in Table 2 are local in all their arguments, and so are most nouns, verbs, etc. The classical example of Partee involves a modal, intransitive verb: if

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

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

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

then $\mathsf{rises}$ 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 $\mathsf{rises}$ closest to our intuitions (and get the right truth value in the example) by setting

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

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.[18]

**4.8. Coindexing; rendering directly into $\mathsf{L}^{\lambda}_{\mathbf{ar}}$.** Roughly speaking, coindexing 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.[19] It is essentially part of the rendering operation, since whether and how it should be done is determined by the informal context discussed in 1.1. It is possible however to construe part of it as a formal operation on terms, to be performed after a preliminary formalization which leaves the indexicals alone. Consider the following examples in the $\lambda$-calculus, that we have already seen:

(30a) $\text{John loves himself} \xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{himself})$
$$\xrightarrow{\text{coindex}}_{\lambda} \Big(\lambda(\jmath)\mathsf{loves}(\jmath, \jmath)\Big)(\mathsf{John})$$

(30b) $\text{John kissed his wife} \xrightarrow{\text{formalize}} \mathsf{kissed}(\mathsf{John}, \mathsf{wife}(\mathsf{his}))$
$$\xrightarrow{\text{coindex}}_{\lambda} \Big(\lambda(\jmath)\mathsf{kissed}(\jmath, \mathsf{wife}(\jmath))\Big)(\mathsf{John})$$

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

[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 4.6:

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

The correct reading is not a matter of logic but one of language, and so it must be done at the rendering stage.

[19] Coindexing is partly determined by the syntactic structure of a given language. The most famous principles forcing, forbidding or allowing coindexing without forcing it are conditions A, B and C of Chomsky's binding theory, into which we cannot go here, so we will confine ourselves to treating some examples.

(30c) John loves his wife and he honors her

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

Symbolically,

$$\xrightarrow{\text{render}} = \xrightarrow{\text{formalize}} + \xrightarrow{\text{coindex}}_1 + \cdots + \xrightarrow{\text{coindex}}_k .$$

We will not attempt to define precisely this operation here, since it is not clear at this point how to do it in full generality.

One thing worth noticing, however, is that *the recursion construct provides an alternative way to co-index* which, in fact, *leads to essentially new renderings* of simple English sentences. For these examples:

(31a) John loves himself $\xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{himself})$
$$\xrightarrow{\text{coindex}}_{\text{ar}} \mathsf{loves}(j, j) \text{ where } \{j := \mathsf{John}\}$$

(31b) John kissed his wife $\xrightarrow{\text{formalize}} \mathsf{kissed}(\mathsf{John}, \mathsf{wife}(\mathsf{his}))$
$$\xrightarrow{\text{coindex}}_{\text{ar}} \mathsf{kissed}(j, \mathsf{wife}(j)) \text{ where } \{j := \mathsf{John}\}$$

(31c) John loves his wife and he honors her

$$\xrightarrow{\text{formalize}} \mathsf{loves}(\mathsf{John}, \mathsf{wife}(\mathsf{his})) \,\&\, \mathsf{honors}(\mathsf{he}, \mathsf{her})$$
$$\xrightarrow{\text{coindex}}_{\text{ar}} \mathsf{loves}(j, \mathsf{wife}(j)) \,\&\, \mathsf{honors}(j, \mathsf{her}) \text{ where } \{j := \mathsf{John}\}$$
$$\xrightarrow{\text{coindex}}_{\text{ar}} \Big(\mathsf{loves}(j, \dot{w}) \,\&\, \mathsf{honors}(j, \dot{w}) \text{ where } \{\dot{w} := \mathsf{wife}(j)\}\Big) \text{ where } \{j := \mathsf{John}\}$$

(31d) $$\approx \mathsf{loves}(j, \dot{w}) \,\&\, \mathsf{honors}(j, \dot{w}) \text{ where } \{\dot{w} := \mathsf{wife}(j), j := \mathsf{John}\}$$

It will turn out that, naturally enough, (30a) and (31a) are referentially synonymous, but (30b) is not referentially synonymous with (31b), and neither is (30c) referentially synonymous with (31c) or its synonym (31d), which we have included for clarity. Moreover, we will show in 9.2 that *the terms in* (31b), *and* (31c) *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 9.3.

**4.9. 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,

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

Here Montague interprets "$\mathsf{John}$" by the evaluation function,[20]

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

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 4.11. We do 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 logical meaning*. Specifically, we will show in 9.4 that the Montague renderings

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

of the classic Frege example are not referentially synonymous, as, of course, they should be.[21] With the typing we have adopted, for any two terms $A, B : \sigma$,

$$A = B \xrightarrow{\text{render}} =_\sigma (A, B) \approx\!\!= =_\sigma (B, A) \equiv \text{rendering(B=A)},$$

simply because the identity relation is symmetric.

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

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

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

$$A\ \phi\ B$$

[20] The evaluation imbedding $x \mapsto \lambda(p)p(x)$ of a set $X$ into its "second dual" is used in many parts of mathematics, most famously in the proof that the space of all bounded, linear functionals on a Hilbert space $H$ is isomorphic with $H$. Its ultrafilter version, $x \mapsto \mathcal{U}(x) = \{U \subseteq X \mid x \in U\}$ is the key to the Stone Representation Theorem for Boolean algebras.

[21] We assume 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.

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.

The discussion reiterates our insistence that *meaning* (intuitively understood) *must be seriously considered in the rendering process*—simply "getting the right denotation" is not enough.

**4.10. Relative clauses.** The same problem of typical ambiguity crops up in the treatment of relative clauses, e.g.,

[Mary (who loves him)] suffers  [every (woman (who loves him))] suffers,

where we have indicated the correct parsing. Here "Mary, who loves him" must be rendered in type $\tilde{\mathsf{e}}$, so that it will be an appropriate argument to "suffers", while "woman, who loves him" must be rendered in type $\tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ so that it can serve as an argument to "every". We assume a constant "$\mathsf{who}$" interpreted by[22]

$$\mathrm{who}(u)(x)(a) = \begin{cases} u(a), & \text{if } x(u,a), \\ er, & \text{otherwise,} \end{cases} \qquad (u : \tilde{\mathsf{e}},\ x : (\tilde{\mathsf{e}} \to \tilde{\mathsf{t}})),$$

and then we follow the method of 4.9.

**(A)** If $X \xrightarrow{\text{render}} \overline{X} : \tilde{\mathsf{e}}$ and $P \xrightarrow{\text{render}} \overline{P} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$, set

$X$ who $P \xrightarrow{\text{render}} \mathsf{who}(\overline{X})(\overline{P})$;

**(B)** If $R \xrightarrow{\text{render}} \overline{R} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$ and $P \xrightarrow{\text{render}} \overline{P} : \tilde{\mathsf{e}} \to \tilde{\mathsf{t}}$, set

$R$ who $P \xrightarrow{\text{render}} \lambda(u)\overline{P}(\mathsf{who}(u, \overline{R}))$.

Now

Mary, who loves him $\xrightarrow{\text{render}} \mathsf{who}(\mathsf{Mary}, \lambda(u)\mathsf{loves}(u, \mathsf{him}))$

yields (in a given state) Mary if she loves him and *er* if she does not, while

woman who loves him $\xrightarrow{\text{render}} \lambda(u)\mathsf{woman}(\mathsf{who}(u)(\lambda(v)\mathsf{loves}(v, \mathsf{him})))$

expresses the relation of being a woman such that she loves him.[23] Putting this together with the quantifier renderings of 4.9 and coindexing "John" with "him",

[22]Notice the interpretation when the entity is not rigid and the property is not local: "the temperature, which is rising" will evaluate to the(temperature)($a$) in a state $a$ in which the temperature is rising, and to *er* in a state in which the temperature is not rising.

[23]To see how errors are treated with this interpretation of the relative clause:

(1) "Mary is a woman who loves him" is true if Mary loves him and evaluates to error of she does not.

(2) "George is a woman who loves him" is false if George loves him and evaluates to error if he does not.

Perhaps this interpretation of the relative clause when the presupposition fails is too naive but, at least, it has the virtue that "$X$ is $R$ who is $P$" is not synonymous with "$X$ is $R$ and $X$ is $P$"; it is a predication, for one thing, not a conjunction.

we get the none-too-simple

John loves some woman who loves him

$$\xrightarrow{\text{render}} \mathsf{some}\Big[\lambda(u)\mathsf{woman}(\mathsf{who}(u)(\lambda(v)\mathsf{loves}(v,j)))\Big](\lambda(v)\mathsf{loves}(j,v)) \quad \text{where } \{j := \mathsf{John}\}$$

**4.11. Coordination.** The phrase

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.[24] Similarly,

The temperature is 90° and rising

(a predication) is not synonymous with

The temperature is 90° and (it) is rising,

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 last three sections, however, it may be worth listing here some alternative renderings directly into $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, which use the recursion construct and (in some cases) produce substantially simpler meanings.[25]

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

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

Thus

John and Mary $\xrightarrow{\text{render}} \lambda(r)(r(x_1)\,\&\,r(x_2))$ 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

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

so that

the teacher and every student

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

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

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

**(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 $90^\circ$ and rising

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

**5. Overview of referential intension theory.** Before we embark on the technicalities in the next two sections, we briefly map the territory.

**5.1. The reduction calculus (Section 6).** We will define a binary relation $\Rightarrow$ of *reduction* between denotational terms, so that, intuitively,

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

The disjunction is needed partly because congruent terms presumably express their meaning equally directly, but also and more importantly, because some terms (e.g., variables) cannot be assigned meanings, but it is still useful to have the reduction calculus still apply to them. We set

(32) $A$ is irreducible $\iff$ for all $B$, if $A \Rightarrow B$, then $A \equiv_c B$.

Meaningful irreducible terms express their meaning directly.

The reduction relation is reflexive and transitive,

$$A \Rightarrow A, \quad \text{if } A \Rightarrow B \text{ and } B \Rightarrow C, \text{ then } A \Rightarrow C,$$

and it is compositional:

**5.1.1. Compositionality for reduction.** *Suppose $A$ is a term, $x$ is a variable or constant of type $\sigma$, and $C_1, C_2$ are terms of type $\sigma$ which are free for $x$ in $A$; then*

$$\text{if } C_1 \Rightarrow C_2, \text{ then } A\{x :\equiv C_1\} \Rightarrow A\{x :\equiv C_2\}.$$

Next will come the following basic result of the theory:

**5.1.2. Canonical Form Theorem.** *For each term $A$, there is a recursive, irreducible term*

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

*such that each $A_i$ is explicit and $A \Rightarrow \mathrm{cf}(A)$; moreover,* $\mathrm{cf}(A)$ *is the unique* (up to congruence) *irreducible term to which $A$ can be reduced, i.e.,*

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

We call $\mathrm{cf}(A)$ the *canonical form of* $A$, and we write

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

If $A$ is explicit and irreducible, then

$$\mathrm{cf}(A) \equiv A \equiv A \text{ where } \{\ \},$$

the last by the convention (16) which allows us to think of all canonical forms as recursive terms.

The terms $A_0, A_1, \ldots, A_n$ are the *parts* of $A$, and $A_0$ is its *head*. The number $n$ (one less than the number of parts) is the *dimension* of $A$, an important invariant of terms determined by their canonical forms. An explicit, irreducible term $A$ has dimension 0; it has only one part $A$, which is also its head.

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

**5.1.3. 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

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

This stands for synonymy on the basis of logical form alone. It will be an immediate consequence of the definition of canonical forms and the properties of the reduction relation that

(35) if $A \Rightarrow B$, then $A \approx_s B$.

**5.2. Referential intensions (Section 7).** Variables and some very simple, *immediate terms* have no meaning, they refer *immediately*. Constants, on the other hand, refer *directly*, but they have meanings, albeit trivial ones which are exhausted by their denotations. Constants contribute differently to the meanings of the terms in which they occur than variables.

The distinction between immediate and direct reference is a central feature of this theory and we will discuss it in 6.3, where immediate terms are defined, and in Section 10, especially 10.6.

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

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

then the *referential intension* $\mathrm{int}(A)$ of $A$ is (intuitively) the abstract algorithm which computes for each valuation g the denotation $\mathrm{den}(A)(\mathrm{g})$, as that was

described in Case (D4) of 3 and the example following it. The precise definition is given in Section 7.

The parts $A_0, A_1, \dots, A_n$ of a term $A$ are *explicit, irreducible terms* which refer directly; 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 (extracted from the syntactic form of $A$) by which $\mathrm{den}(A)(\mathrm{g})$ is computed from them.

**5.2.1. Canonical forms and truth conditions.** If $A$ is a Carnap intension, meaning that $A : \tilde{\mathsf{e}}$, 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 our approach to language 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 very possibility Davidson doubts.

**5.3. Referential and logical synonymy (Section 8).** Two proper terms are *referentially synonymous* if their referential intensions (in our universe) 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 valuations of the variables. We will be using the notation

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

The precise, general definition of natural isomorphism in 7.5 is a bit technical, but it implies a very simple characterization of the referential synonymy relation on terms:[26]

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

$$A \Rightarrow_{\mathrm{cf}} A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}, \tag{36a}$$

$$B \Rightarrow_{\mathrm{cf}} B_0 \text{ where } \{\dot{p}_1 := B_1, \dots, \dot{p}_n := B_n\}, \tag{36b}$$

*for some $n \geq 0$ and suitable $A_0, A_1, \dots, A_n, B_0, B_1, \dots, B_n$, so that the following two conditions hold:*

[26] Condition (RS1) in this result is a mild, technical refinement of the notions of referential intension and synonymy that were used in Moschovakis [2006]. It does not affect the examples from natural language.

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

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

(RS2) *For our universe $\mathfrak{M}_0$,*

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

The result reduces referential synonymy to a trivial check of free variable occurrences and a system of (effectively determined) denotational identities between explicit, irreducible terms. It is at the heart of the proposed theory of meaning. Notice that $n = 0$ is allowed in this theorem (by the convention (16)) and occurs when $A$ and $B$ are explicit, irreducible terms: such terms are synonymous exactly when they have the same free variables and the same denotation.

**5.3.2. Logical synonymy.** Referential synonymy differs from syntactic synonymy (introduced in Section 5.1.3) in that it takes into account the interpretation of the constants in our universe as well as the constructs of $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$. For example,

$$\mathsf{Charles\ Lutwidge\ Dodgson} \approx \mathsf{Lewis\ Carroll},$$

because these are explicit, irreducible terms whose denotations determines their referential intensions—and they both denote (rigidly) the same author.

*Logical synonymy* is intermediate between syntactic and referential synonymy: it takes into account the meaning of the constructs of $\mathsf{L}^{\lambda}_{\mathrm{ar}}(K)$ but not the particular structure in which we interpret it. Two terms are *logically synonymous* if they satisfy (36a), (36b) with suitable $A_i, B_i$ so that

(LS1) The corresponding parts $A_i, B_i$ of $A$ and $B$ have the same free variables, i.e., for every variable $x$ of either kind,

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

(LS2) For every structure $\mathfrak{M}$ and every $i = 0, \dots, n$, $\mathfrak{M} \models A_i = B_i$.

We write

$$A \approx_{\ell} B \iff A \text{ is logically synonymous with } B.$$

For example,

$$\lambda(u, v)\mathsf{love}(u, v) \approx_{\ell} \mathsf{love},$$

because these are irreducible terms with just one part,

$$\lambda(u, v)\mathsf{love}(u, v) \Rightarrow_{\mathrm{cf}} \lambda(u, v)\mathsf{love}(u, v) \text{ where } \{\ \}, \quad \mathsf{love} \Rightarrow_{\mathrm{cf}} \mathsf{love} \text{ where } \{\ \},$$

and by $\beta$-reduction, no matter what the love relation in a structure $\mathfrak{M}$,

$$\mathfrak{M} \models \lambda(u, v)\mathsf{love}(u, v) = \mathsf{love}.$$

It is immediate from the definitions and the Referential Synonymy Theorem that

$$(37) \quad A \equiv_c B \Longrightarrow A \approx_s B \Longrightarrow A \approx_{\ell} B \Longrightarrow A \approx B,$$

and none of these implications can be reversed, by the examples above.

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

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

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

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

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

$$\frac{\models C = D}{C \sim D}\,(*) \quad \text{hence:} \quad \frac{}{\Big(\lambda(u)C\Big)(v) \sim C\{u :\equiv v\}}\,(**)$$

$(*)$ : $C, D$ are both explicit and irreducible, with the same free variables
For referential synonymy $\approx$:
$\models C = D \iff$ for all valuations g, $\mathfrak{M}_0, \mathrm{g} \models C = D$
For logical synonymy $\approx_\ell$:
$\models C = D \iff$ for all $\mathfrak{M}$ and all valuations g, $\mathfrak{M}, \mathrm{g} \models C = D$
$(**)$ $u$ and $v$ are pure variables, and the substitution $C\{u :\equiv v\}$ is free

TABLE 4. The calculi of referential and logical synonymy.

We are most interested in establishing real synonymies and non-synonymies in our universe, of course; but most of the proofs depend on recognizing (easily) syntactic or logical synonymies, and then finishing the argument with some appeal to empirical facts about our universe.

**5.3.3. The calculi of referential and logical synonymy.** *Referential and logical synonymy satisfy the axioms and rules of inference in Table 4.*

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 synonymy.

It is also worth noting the following consequence of these calculi:

**5.3.4. Compositionality Theorem.** *For all terms $A, B, C$ and every variable $x$ such that* $\mathrm{type}(x) = \mathrm{type}(B) = \mathrm{type}(C)$*, if $\sim$ is either $\approx_\ell$ or $\approx$,*

$$\text{if } B \sim C, \text{ then } A\{x :\equiv B\} \sim A\{x :\equiv C\},$$

*assuming that the substitutions are free.*

**5.3.5. The proof systems $\mathsf{Syn}$ and $\mathsf{Syn}_l$; completeness results.** Let $\mathsf{Syn}$ be the proof system constructed by adding to the logical calculus for $\sim$ in Table 4

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

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

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

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

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

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

TABLE 5. The reduction calculus: congruence, transitivity, compositionality.

as an axiom every instance of $A \sim B$ such that

either $A \Rightarrow B$ or $A, B$ are explicit, irreducible and $\mathfrak{M}_0 \models A = B$. (Syn)

Similarly, let $\mathsf{Syn}_l$ be the proof system constructed by adding every instance of $A \sim B$ such that

either $A \Rightarrow B$ or $A, B$ are explicit, irreducible

and $\mathfrak{M} \models A = B$ for every $\mathfrak{M}$. ($\mathsf{Syn}_l$)

Then: Syn *is complete for referential synonymy and* $\mathsf{Syn}_l$ *is complete for logical synonymy.*

These results are about as far as we can go in axiomatizing the synonymy relations, and both of them have limitations.

The set of axioms of Syn is undecidable if we admit an arbitrary, infinite set $K$ of constants, and it may be undecidable for arbitrary, finite set $K$, even if we insist that they are denotational. This is an important, open problem in the theory.

On the other hand, the set of axioms of $\mathsf{Syn}_l$ is decidable, by classical results in the $\lambda$-calculus, cf. Tait [1967], Friedman [1974]; but the "arbitrary structures" $\mathfrak{M}$ in its formulation include those in which the set of truth values $\mathbb{T}_\mathsf{t}$ is infinite, which are quite far from our universe—at least as it is usually conceived.

Still, these proof systems are very useful in practice, because the axioms that we need to establish specific referential or logical synonymies are often obvious, or can be established easily by applying $\beta$-conversion.

In many cases, we can apply the definitions and theorems in this Section 5 as "black boxes", to derive synonymies and non-synonymies without the detailed development in the sequel. We will establish them first for the denotational part of the system, and then extend them in Section 11 to the full language with attitudinal constants.

**6. The reduction calculus.** The reduction relation $A \Rightarrow B$ between terms is determined by ten rules listed in the four Tables 5 – 8; it is the smallest binary relation on terms which satisfies these ten rules. We discuss and explain them in turn.

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

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

TABLE 6. The reduction calculus: rules for recursion.

**6.1. Congruence, transitivity, compositionality.** The first five rules are listed in Table 5, 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.

**6.2. The reduction rules for recursion.** These are listed in Table 6, and they allow us to combine recursive definitions. In stating them we have used the abbreviations

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

where it is assumed that $\dot{p}_1, \dots, \dot{p}_n, \dot{q}_1, \dots \dot{q}_m$ are distinct locations and we have omitted some (mostly obvious) restrictions on occurrences of variables. Here they are in full, with some examples.

**6.2.1. The head-rule (head).** *Restriction*: *No $\dot{p}_i$ occurs free in any $B_j$.*[27] The rule allow us to bring the head term into the system of assignments:

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

**6.2.2. The Bekič-Scott rule (B-S).** *Restriction*: *No $\dot{q}_j$ occurs free in any $A_i$.* Example:

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

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.

[27] 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 we will not bring it up again.

**6.2.3. The recursion-application rule (recap).** *Restriction*: *No $\dot{p}_i$ occurs free in $B$.* For an example, let[28]

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

a term which is logically synonymous with

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 $x$,

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

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

*Procedure* 1.

Stage 1: Set $\dot{h} := \mathrm{He}$, $\dot{s} := \mathrm{Scott}$.

Stage 2: Set $a := \mathrm{g}(x)$; if $\dot{h}(a) = \dot{s}(a) = \text{Sir Walter}$, 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 $\dot{a} := \mathrm{g}(x)$.

Stage 2′: $\dot{h}' := \mathrm{He}(a)$, $\dot{s}' := \mathrm{Scott}(a) = \text{Sir Walter}$.

Stage 3′: If $\dot{h}' = \dot{s}' = \text{Sir Walter}$, 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!

**6.3. Immediate terms.** The *immediate applicative terms* are those of the form

$$E :\equiv u \mid \dot{p} \mid x(v_1, \dots, v_n) \mid \dot{p}(v_1, \dots, v_n) \qquad \text{(Immediate applicative terms)}$$

where $x, u, v_1, \dots, v_n$ are pure variables and $\dot{p}$ is a recursion variable (and the types match, of course, so they are terms). The *immediate $\lambda$-terms* are those of the form

$$X :\equiv \lambda(v_1, \dots, v_n)(E) \qquad \text{(Immediate } \lambda\text{-terms)}$$

where $v_1, \dots, v_n$ are pure variables and $E$ is applicative immediate.

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

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

TABLE 7. The reduction calculus: the application rule.

The immediate terms comprise the immediate applicative and $\lambda$-terms together.

The key point is that if $x$ is a variable of either kind and $u$ is pure, then $x(u)$ is immediate; but a recursion variable cannot be the argument of an application in an immediate term.

Terms which are not immediate are *proper*.

We can think of immediate terms as *generalized variables* of sorts, and as we will see, they share many properties with variables.

Constants are proper, in any type. To understand this, don't think of Scott, whose "computation" appears to be trivial, but (as in Footnote 11 above) think of the number $\pi$, whose exact computation requires an infinite number of steps. It is standard advice to beginning programmers to set up an assignment $\dot{p} := \pi$ and then replace "$\pi$" by "$\dot{p}$" throughout their program, if "$\pi$" occurs many times. The new program expresses a more efficient algorithm, which computes (some reasonable approximation to) $\pi$ only once, stores the value, and then just reads it each time it is needed.

**6.4. The application rule (ap).** This is stated in Table 7. To understand why the non-immediacy restriction is needed, consider the example

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

If the unrestricted rule preserved meanings, we would have

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

and then we could continue with the reductions

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

so that, in particular,

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

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

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

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 we do not allow (at this stage) the interpretation of constants 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 $A : \mathsf{t}$,

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

because $\dot{p} : \mathsf{t}$ 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.

**6.4.1. The canonical form of "John loves Mary".** The last (still missing) reduction rule does not affect $\lambda$-free terms, and so

$$\mathsf{runs}(j) \text{ where } \{j := \mathsf{John}\}$$

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

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

of the (ap) rule gives us the canonical form of $\mathsf{runs}(\mathsf{John})$, if we assume for a moment the Canonical Form Theorem 5.1.2, i.e., that $\mathsf{runs}(\mathsf{John})$ reduces to a unique irreducible term up to congruence. Let us write out one more complete reduction to canonical form which is just as trivial but illustrates the use of the recursion rules:

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

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

In the same way, we can compute

(38) He is Scott $\Rightarrow_{\mathrm{cf}} (\dot{h} = \dot{s})$ where $\{\dot{h} := \mathsf{He}, \dot{s} := \mathsf{Scott}\}$,

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

(39) $1 + 5 = 2 \times 3$

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

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

where for $i = 1, \dots, n$, $\dot{p}'_i$ is a fresh recursion variable and $A'_i$ is defined by the replacement

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

TABLE 8. The $\lambda$-rule.

**6.5. The $\lambda$-rule.** To motivate the $\lambda$-rule in Table 8, 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:

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

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

$\dot{w} := \text{wife}(u)$

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

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

which is what the $\lambda$-rule allows us to do.

**6.5.1. The canonical form of "Every man danced with his wife".** If wife is a constant, then

$\lambda(u)\mathsf{wife}(u) \approx_\ell \mathsf{wife}$,

and so we can conclude (changing the variable names and appealing to Theorem 5.3.3) that

$\lambda(u)(B) \Rightarrow_{\text{cf}} \lambda(u)\mathsf{danced}(u, \dot{w}(u)) \text{ where } \{\dot{w} := \lambda(u)\mathsf{wife}(u)\}$
$\approx_\ell \lambda(u)\mathsf{danced}(u, \dot{w}(u)) \text{ where } \{\dot{w} := \mathsf{wife}\}$.

If, however, wife is an abbreviation introduced by

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

then the computation continues:

$$\begin{aligned}\lambda(u)\mathsf{wife}(u) &\Rightarrow \lambda(u)(\mathsf{the}(\lambda(v)\mathsf{married}(u,v))) && \text{(ap)}\\ &\Rightarrow \lambda(u)[\mathsf{the}(\dot m)\ \mathsf{where}\ \{\dot m := \lambda(v)\mathsf{married}(u,v)\}] && (\lambda\text{-rule})\\ &\Rightarrow \lambda(u)\mathsf{the}(\dot m'(u))\ \mathsf{where}\ \{\dot m' := \lambda(u)\lambda(v)\mathsf{married}(u,v)\},\end{aligned}$$

and without noting any more the (ap) or recursion rules used,

$$\begin{aligned}&\lambda(u)(B) \Rightarrow \lambda(u)\mathsf{danced}(u,\dot w(u))\ \mathsf{where}\ \{\dot w := \lambda(u)\mathsf{wife}(u)\}\\ &\quad\Rightarrow \lambda(u)\mathsf{danced}(u,\dot w(u))\ \mathsf{where}\ \{\dot w := \lambda(u)\mathsf{the}(\lambda(v)\mathsf{married}(u,v))\\ &\quad\Rightarrow \lambda(u)\mathsf{danced}(u,\dot w(u))\ \mathsf{where}\\ &\qquad\qquad \Big\{\dot w := \lambda(u)\mathsf{the}(\dot m'(u))\ \mathsf{where}\ \{\dot m' := \lambda(u)\lambda(v)\mathsf{married}(u,v)\}\Big\}\\ &\quad\Rightarrow_{\mathrm{cf}} \lambda(u)\mathsf{danced}(u,\dot w(u))\\ &\qquad\qquad\qquad \mathsf{where}\ \{\dot w := \lambda(u)\mathsf{the}(\dot m'(u)), m' := \lambda(u)\lambda(v)\mathsf{married}(u,v)\}.\end{aligned}$$

Finally, if we assume now that married is a constant, we have

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

and substituting this in the canonical form (and changing variable names again), we get the relatively simple logical synonymy

$$\begin{aligned}&\lambda(u)(\mathsf{danced}(u),\mathsf{wife}(u))\\ &\qquad \approx_\ell \lambda(u)\mathsf{danced}(u,\dot w(u))\ \mathsf{where}\ \{\dot w := \lambda(u)\mathsf{the}(\dot m(u)), \dot m := \mathsf{married}\},\end{aligned}$$

granting the abbreviation (40).

From this, we get with some additional computation:

Every man danced with his wife

$$\begin{aligned}&\xrightarrow{\text{render}} \mathsf{every}(\mathsf{man})(\lambda(u)(\mathsf{danced}(u),\mathsf{wife}(u)))\\ &\Rightarrow \mathsf{every}(\mathsf{man})(\dot d)\ \mathsf{where}\ \{\dot d := \lambda(u)(\mathsf{danced}(u),\mathsf{wife}(u))\}\\ &\Rightarrow \mathsf{every}(\dot M)(\dot d)\ \mathsf{where}\ \{\dot M := \mathsf{man}, \dot d := \lambda(u)(\mathsf{danced}(u),\mathsf{wife}(u))\}\\ &\Rightarrow \mathsf{every}(\dot M)(\dot d)\ \mathsf{where}\ \{\dot M := \mathsf{man},\\ &\qquad\qquad\qquad\qquad \dot d := \lambda(u)\mathsf{danced}(u,\dot w(u)),\\ &\qquad\qquad\qquad\qquad \dot w := \lambda(u)\mathsf{the}(\dot m'(u)),\\ &\qquad\qquad\qquad\qquad \dot m' := \lambda(u)\lambda(v)\mathsf{married}(u,v)\}.\end{aligned}$$

The last term is the canonical form of "Every man danced with his wife".

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

**6.6. Reduction and equality.** *If $A \Rightarrow B$, then for every structure $\mathfrak{M}$,*

$$\mathfrak{M} \models A = B.$$

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

It is clear from the examples that the explicit irreducible terms play a very special role in the reduction calculus. They can get rather complex, but they admit a simple, general characterization.

**6.7. Characterization of irreducible terms.** (a) *Constants and immediate terms are irreducible.*

(b) *An application term $A(B)$ is irreducible if and only if $B$ is immediate and $A$ is explicit and irreducible.*

(c) *A $\lambda$-term $\lambda(u)(A)$ is irreducible if and only if $A$ is explicit and irreducible.*

(d) *A recursive term $A_0$ where $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$ is irreducible if and only all the parts $A_0, \dots, A_n$ are explicit and irreducible.*

PROOF is simple, by inspection of the reduction rules. ⊣

Some examples:

$$\mathsf{c}(\lambda(u)\dot{p}(u,v)), \quad \lambda(v)(\mathsf{c}(\lambda(u)\dot{p}(u,v))(\dot{q}(v,z)), \quad \lambda(z)(\mathsf{c}(\lambda(u)\dot{p}(u,v))(\dot{q}(v,z)))(y(x,x)) \tag{41}$$

Notice that if we simplify these terms using $\beta$ reduction, they are no longer irreducible: for example,

$$\lambda(v)(\mathsf{c}(\lambda(u)\dot{p}(u,v))(\dot{q}(v,z)) = \mathsf{c}(\lambda(u)\dot{p}(u,\dot{q}(v,z)) \qquad \text{(by } \beta\text{-reduction)}$$

and the term on the right is no longer irreducible, because $\lambda(u)\dot{p}(u,\dot{q}(v,z)$ is not immediate. It goes without saying that complex, irreducible terms like these do not come up naturally in simple examples from natural language.

**6.8. 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 $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$ $(n \geq 0)$. If $X$ is immediate, then

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

and if $B$ is proper and $\mathrm{cf}(B) \equiv B_0$ where $\{\dot{q}_1 := B_1, \dots, \dot{q}_n := B_m\}$, then

$$\mathrm{cf}(A(B)) :\equiv A_0(\dot{q}_0) \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n, \dot{q}_0 := B_0, \dot{q}_1 := B_1, \dots, \dot{q}_m := B_m\}.$$

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

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\} \qquad (n \geq 0),$$

then,

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

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

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

(CF4) If $A \equiv A_0$ where $\{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}$ with $n \geq 0$ and if, for $i = 0, \dots, n$,

$$\mathrm{cf}(A_i) \equiv A_{i,0} \text{ where } \{\dot{p}_{i,1} := A_{i,1}, \dots, \dot{p}_{i,k_i} := A_{i,k_i}\} \qquad (k_i \geq 0),$$

then

$$\begin{aligned}\mathrm{cf}(A) :\equiv A_{0,0} \text{ where } \{ \quad & \dot{p}_{0,1} := A_{0,1}, \dots, \dot{p}_{0,k_0} := A_{0,k_0},\\ \dot{p}_1 := A_{1,0}, & \dot{p}_{1,1} := A_{1,1}, \dots, \dot{p}_{1,k_1} := A_{1,k_1},\\ & \vdots\\ \dot{p}_n := A_{n,0}, & \dot{p}_{n,1} := A_{n,1}, \dots, \dot{p}_{n,k_n} := A_{n,k_n}\}.\end{aligned}$$

The next result provides in particular a proof of the Canonical Form Theorem 5.1.2.

**6.8.1. Basic properties of canonical forms.** *For every term $A$:*

(1) *The canonical form of $A$ is a term*

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

*with explicit, irreducible parts $A_0, A_1, \dots, A_n$, so that it is irreducible. A constant $\mathsf{c}$ or a variable $x$ of either kind 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 6.7 and then using the definition of $\mathrm{cf}(A)$ if $A$ is recursive. 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). ⊣

**6.9. Proofs of 5.3.3 – 5.3.5.** These facts are also established by somewhat messy, long inductions, using the results of these section, the Referential Synonymy Theorem 5.3.1, the definition of logical synonymy and the implications (37). The following facts are also useful in these arguments:

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

(2) *If $z : \sigma$ is a constant $\mathsf{c}$, 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*

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

The first of these is trivial and the second is proved by induction on $A$; it is a basic fact which should be established first.

**7. Referential intensions.** We define here the referential intension $\mathrm{int}(A)$ of a proper term and derive its basic properties.

The material in this section is not important for most of the applications, if one is willing to assume the Referential Synonymy Theorem 5.3.1 or simply take

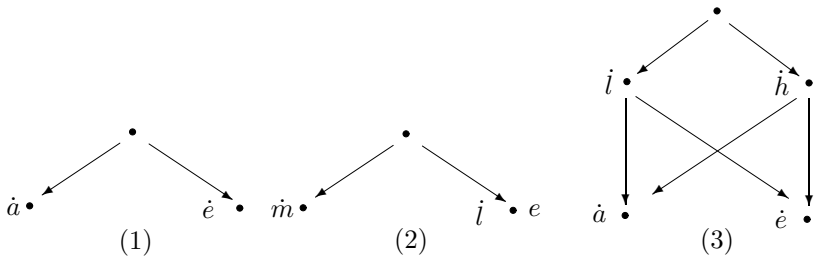



FIGURE 1. Shapes.

its statement as a definition of referential synonymy. We will use it, however, in Section 11 on propositional attitudes.

**7.1. Basic definition; shapes.** Suppose that $A$ is a proper (non-immediate) term with canonical form

$$\mathrm{cf}(A) \equiv A_0 \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\} \qquad (n \geq 0).$$

The *shape* of $A$ is the tuple

$$\text{(42a)} \quad \mathrm{shape}(A) = (\dot{p}_1, \dots, \dot{p}_n, \vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0, \dots, \vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)$$

where $\vec{\mathbf{f}}_i$ is a listing of the variables (of either kind) which occur free in both $A$ and $A_i$ and $\vec{\mathbf{r}}_i$ is a listing of those variables in the list $\dot{p}_1, \dots, \dot{p}_n$ which occur in $A_i$. We can picture $\mathrm{shape}(A)$ as a *labelled, acyclic directed graph* on the vertices $\dot{p}_0, \dot{p}_1, \dots, \dot{p}_n$, with a *head vertex* $\dot{p}_0$ added to the locations of $\mathrm{cf}(A)$, the edge relation

$$\dot{p}_i \to \dot{p}_j \iff \dot{p}_j \text{ occurs (free) in } A_i,$$

and each vertex $\dot{p}_i$ labelled with the list $\vec{\mathbf{f}}_i$.

For example,

$$\mathrm{shape}(\mathsf{loves}(\dot{a}, \dot{e}) \text{ where } \{\dot{a} := \mathsf{Abelard}, \dot{e} := \mathsf{Eloise}\}) = (\dot{a}, \dot{e}, \emptyset, \langle \dot{a}, \dot{e} \rangle, \emptyset, \emptyset, \emptyset, \emptyset);$$

this is the graph (1) in Figure 1. If

$$\begin{aligned} \text{Every man loves } e \xrightarrow{\text{render}} A &\equiv \mathsf{every}(\mathsf{man})(\lambda(x)\mathsf{loves}(x, e)) \\ &\Rightarrow_{\mathrm{cf}} \mathsf{every}(\dot{m})(\dot{l}) \text{ where } \{\dot{m} := \mathsf{man}, \dot{l} := \lambda(x)\mathsf{loves}(x, e)\}, \end{aligned}$$

then $\mathrm{shape}(A) = (\dot{m}, \dot{l}, \emptyset, \langle \dot{m}, \dot{l} \rangle, \emptyset, \emptyset, \langle e \rangle, \emptyset)$, pictured in (2) of Figure 1. The last graph (3) in the figure is the shape of the term

$$\dot{l} \text{ and } \dot{h} \text{ where } \{\dot{l} := \mathsf{loved}(\dot{a}, \dot{e}), \dot{h} := \mathsf{honored}(\dot{a}, \dot{e}), \dot{a} := \mathsf{Abelard}, \dot{e} := \mathsf{Eloise}\}$$

which is the canonical form of "Abelard loved and honored Eloise".

The shape of a term $A$ determines the number of parts in its canonical form and the (putative) dependence of each of the recursion variables in $\mathrm{cf}(A)$ on the

others and on the free variables of $A$.[29] It codifies the most important aspects of the syntactic structure of $A$ and it is the first part of $\mathrm{int}(A)$. For the second part, set for $i = 0, \dots, n$

$$\alpha_i(\mathrm{g}, d_1, \dots, d_n) = \mathrm{den}(A_i)(\mathrm{g}\{\dot{p}_1 := d_1, \dots, \dot{p}_n := d_n\})$$

where g is an arbitrary valuation of the variables, and let

(42b) $\mathrm{system}(A) = (\alpha_0, \dots, \alpha_n)$.

This is the system of functions which computes $\mathrm{den}(A)(\mathrm{g})$.

The *referential intension* of $A$ is the pair of its shape and its system,

(42c) $\mathrm{int}(A) = (\mathrm{shape}(A), \mathrm{system}(A))$.

For example,

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

where

$$\begin{aligned}\alpha_0(\mathrm{g}, a, e) &= \mathrm{loves}(a, e), \\ \alpha_1(\mathrm{g}) &= \mathrm{Abelard}, \\ \alpha_2(\mathrm{g}) &= \mathrm{Eloise}.\end{aligned}$$

Notice that if $A, A_0 : \sigma$ and $A_i : \sigma_i$ for $i = 1, \dots, n$, then

(43a) $\alpha_0 : \mathrm{G} \times \mathbb{T}_{\sigma_1} \times \cdots \times \mathbb{T}_{\sigma_n} \to \mathbb{T}_\sigma$,

(43b) for $i = 1, \dots, n$, $\alpha_i : \mathrm{G} \times \mathbb{T}_{\sigma_1} \times \cdots \times \mathbb{T}_{\sigma_n} \to \mathbb{T}_{\sigma_i}$,

(43c) the relation $j \prec i \iff \dot{p}_j \in \vec{\mathbf{r}}_i$ is acyclic,

(43d) if $d_j = d'_j$ whenever $\dot{p}_j \in \vec{\mathbf{r}}_i$
and $\mathrm{g}(x) = \mathrm{g}'(x)$ for every $x \in \vec{\mathbf{r}}(A)$, then
$\alpha_i(\mathrm{g}, d_1, \dots, d_n) = \alpha_i(\mathrm{g}', d'_1, \dots, d'_n)$.

"Acyclic" in (43c) means that we can assign a natural number $\mathrm{rank}(\dot{p}_i)$ to each of the variables $\dot{p}_1, \dots, \dot{p}_n$ so that

$$\dot{p}_j \in \vec{\mathbf{r}}(A_i) \Longrightarrow \mathrm{rank}(\dot{p}_j) < \mathrm{rank}(\dot{p}_i),$$

and it holds because the system $\{\dot{p}_1 := A_1, \dots, \dot{p}_n = A_n\}$ is acyclic.

**7.2. Acyclic recursors.** It is not hard to abstract from (42a) – (42c) and (43a) – (43d) a general definition of acyclic recursors.

An *acyclic recursor* on the set G of variable valuations to $\mathbb{T}_\sigma$ is a pair of tuples

(44) $\alpha = (\mathrm{shape}(\alpha), \mathrm{system}(\alpha))$
$= ((\dot{p}_1, \dots, \dot{p}_n, \vec{\mathbf{f}}_0, \vec{\mathbf{r}}), \dots, \vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n), (\alpha_0, \alpha_1, \dots, \alpha_n))$,

such that the following conditions hold:

[29] The shape of a term is not uniquely defined, because the specific choice of bound variables $\dot{p}_1, \dots, \dot{p}_n$ $\mathrm{cf}(A)$ and the enumeration of the lists of variables $\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i$ are not specified, so we should properly speak of "*a shape*"; but we could easily make these choices of variables and order of listing canonical, and we will not bother with it.

(AR1) $\dot{p}_1, \dots, \dot{p}_n$ are distinct recursion variables of respective types $\sigma_1, \dots, \sigma_n$.
(AR2) $\vec{\mathbf{f}}_i$ and $\vec{\mathbf{r}}_i$ are lists of variables, such that $\vec{\mathbf{r}}_i \subseteq \{\dot{p}_1, \dots, \dot{p}_n\}$ and $\dot{p}_j \notin \vec{\mathbf{f}}_i$.
(AR3) $\alpha_0 : \mathrm{G} \times \mathbb{T}_{\sigma_1} \times \cdots \times \mathbb{T}_{\sigma_n} \to \mathbb{T}_\sigma$.
(AR4) For $i = 1, \dots, n$, $\alpha_i : \mathrm{G} \times \mathbb{T}_{\sigma_1} \times \cdots \times \mathbb{T}_{\sigma_n} \to \mathbb{T}_{\sigma_i}$.
(AR5) The relation $j \prec i \iff \dot{p}_j \in \vec{\mathbf{r}}_i$ is acyclic.
(AR6) If $d_j = d'_j$ whenever $\dot{p}_j \in \vec{\mathbf{r}}_i$ and $\mathrm{g}(x) = \mathrm{g}'(x)$ for every $x \in \vec{\mathbf{f}}_i$, then $\alpha_i(\mathrm{g}, d_1, \dots, d_n) = \alpha_i(\mathrm{g}', d'_1, \dots, d'_n)$.

We call $\sigma_1 \times \cdots \times \sigma_n$ the *internal type* of $\alpha$, $\vec{\mathbf{f}}_0 \cup \cdots \cup \vec{\mathbf{f}}_n$ the set of its *free variables*, and the number $n$ its *dimension*, and we write

$$\alpha = (\mathrm{shape}(\alpha), \mathrm{system}(\alpha)) : \mathrm{G} \rightsquigarrow \mathbb{T}_\sigma.$$

The notation is justified, because $\alpha$ determines (or *computes*) a function

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

as follows:

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

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

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

guaranteed by the acyclicity condition.

A recursor $\alpha = ((\vec{\mathbf{f}}_0), (\alpha_0))$ of dimension 0 is called *trivial*, as it is completely determined by the function $\overline{\alpha} = \alpha_0 : \mathrm{G} \to \mathbb{T}_\sigma$ and a set $\vec{\mathbf{f}}_0$ of its "free variables" on which $\alpha_0(\mathrm{g})$ depends, i.e.,

$$\text{if } \mathrm{g}(x) = \mathrm{g}'(x) \text{ for every } x \in \vec{\mathbf{f}}_0, \text{ then } \alpha_0(\mathrm{g}) = \alpha_0(\mathrm{g}').$$

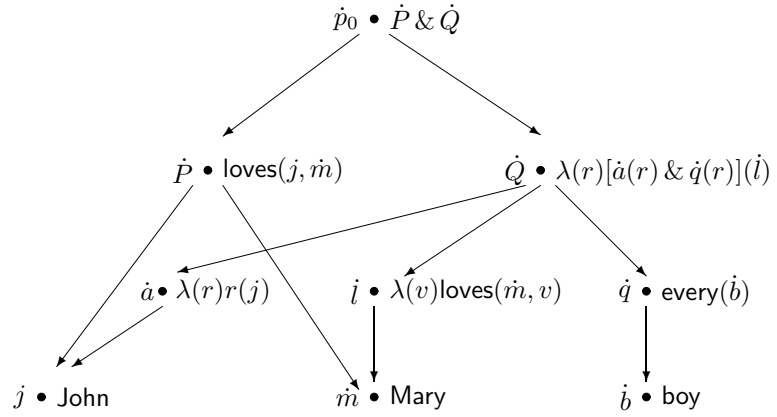
Thus the referential intension of a non-immediate term $A : \sigma$ is an acyclic recursor $\mathrm{int}(A) : \mathrm{G} \rightsquigarrow \mathbb{T}_\sigma$, and by the denotational semantics in Section 3 and Theorem 6.6, for every valuation g,

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

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

**7.3. 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 locations $\dot{p}_1, \dots, \dot{p}_n$ of $\mathrm{cf}(A)$ together with a head node; the part $A_i$ labeling the node $\dot{p}_i$; and an arrow put from $\dot{p}_i$ to $\dot{p}_j$ if $\dot{p}_j$ occurs in $A_i$. This is the same graph which pictures $\mathrm{shape}(A)$, with a more informative labelling which identifies the parts of $A$, and the idea can be used to represent every acyclic recursor by an appropriate labelled graph. Figure 2 pictures this circuit for

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




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

FIGURE 2. *John loves Mary and she loves him and every boy.*

a reasonably complex sentence whose rendering involves both coindexing and coordination.[30]

**7.4. 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" $\mathrm{int}(A)$ as in (42c), basically an annotated 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] in general, and also (briefly) in the last section of Moschovakis [2006], and we will not repeat these arguments here. We claim only that we have constructed a precise model of the basic picture of a Fregean theory of meaning,

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

where $\mathrm{int}(A)$ is an object which *computes* in a precise sense $\mathrm{den}(A)$ and which purports to model the meaning of $A$. Whatever value the construction may have should be determined by its results—the synonymies and non-synonymies it predicts, how closely they agree with our intuitions, and whether the logic of meanings and synonymy that they determine is "the right one".

**7.5. Natural recursor isomorphism.** An acyclic recursor (44) determines for each valuation g the system of mutual recursive equations

[30]The construction of this canonical form assumes a coordination operation on formal terms which is executed after all coindexing operations and which distinguishes immediate from proper arguments, so that

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

$$(45)\quad \begin{cases} d_1 = \alpha_1(\mathrm{g}, d_1, \dots, d_n) \\ \quad\vdots \\ d_n = \alpha_n(\mathrm{g}, d_1, \dots, d_n) \end{cases}$$

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 (45) 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 appropriate equivalence relation is that of *labelled graph isomorphism*, a one-to-one correspondence of the nodes of one labelled graph with those of another which preserves the edge relations and the labels. We also put down, for the record, the explicit, rather technical definition.

Two acyclic recursors

$$\alpha = ((\dot{p}_1, \dots, \dot{p}_n, \vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0, \dots, \vec{\mathbf{f}}_n, \vec{\mathbf{f}}_n), (\alpha_0, \alpha_1, \dots, \alpha_n))$$
$$\beta = ((\dot{q}_1, \dots, \dot{q}_m, ), \vec{\mathbf{f}}'_0, \vec{\mathbf{r}}'_0, \dots, \vec{\mathbf{f}}'_n, \vec{\mathbf{f}}'_n)(\beta_0, \beta_1, \dots, \beta_m))$$

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*, if they have the same dimension ($m = n$) and there is a permutation

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

such that:

(1) $\sigma_{\pi(i)} \equiv \tau_i$ for $i = 1, \dots, n$;
(2) For each $i = 0, \dots, n$, $\vec{\mathbf{f}}'_i = \vec{\mathbf{f}}_{\pi(i)}$ and $\vec{\mathbf{r}}'_i = \langle \dot{q}_{\pi(j)} \mid \dot{p}_j \in \vec{\mathbf{r}}_i\rangle$;
(3) $\alpha_{\pi(i)}(\mathrm{g}, d_1, \dots, d_n) = \beta_i(\mathrm{g}, d_{\pi(1)}, \dots, d_{\pi(n)}) \quad (\mathrm{g} \in \mathrm{G}, d_i \in \mathbb{T}_{\sigma_i}, i = 0, \dots, n)$.

We set

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

**8. Referential and logical 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.[31] In symbols:

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

The Referential Synonymy Theorem 5.3.1 follows immediately from this definition, and it is much easier to understand and apply than chasing natural isomorphisms, as we did in Section 6.9. For most of our purposes here, it might as well be taken as the definition of referential synonymy.

[31] It is quite easy to check that if $X, Y$ are immediate, then

$$\mathfrak{M}_0 \models X = Y \iff X \equiv_c Y,$$

so the definition is a bit simpler than it looks.

**8.1. 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 the calculus of referential intensions, variables (and the more general, immediate terms) behave a little like 0 in the arithmetic of fractions: it is simply not possible to assign any conventional (trivial) value to $\frac{1}{0}$ and still have the usual rules of arithmetic hold.

**9. Non-synonymy.** The Reduction and Synonymy Calculi are very effective tools for establishing simple synonymies; for example, if $\mathrm{type}(A) = \mathrm{type}(B)$, then

$$(A = B) \approx_{\ell} (B = A),$$

since

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

where the crucial, second step is valid because for any structure $\mathfrak{M}$,

$$\mathfrak{M} \models \dot{a} = \dot{b} \iff \dot{b} = \dot{a}.$$

Proofs of non-synonymy, however, are not so simple for complex terms, because the only basic tool we have is to compute the canonical forms, and this can be quite tedious—effective, but tedious. We collect in this section a few interesting non-synonymy results.

**9.1. The unique occurrence property of $\lambda$-calculus terms.** (1) *No recursion variable occurs in more than one part of a $\lambda$-calculus term.*

(2) *Suppose a recursion $\dot{p}$ 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 $\dot{p}$, i.e., for some valuation of the variables* g *and objects $r, r'$,*

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

*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 $\lambda$-calculus terms and using the rules (CF1)–(CF4) in the construction of canonical forms in Section 6.8. For example, looking at (CF4), the only way in which some $\dot{p}_i'$ can occur in $A_k'$ and also in $A_l'$, with $k \neq l$, is if $\dot{p}_i$ 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

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

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

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

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

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

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

**9.2. John can't love and honor his wife properly in the $\lambda$-calculus.** *The terms in* (31b) *and* (31c) *are not referentially synonymous with explicit terms.*

PROOF. If wife is a constant, then the canonical form of (31b) is

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

and the canonical form of (31c) is

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

The location $j$ occurs in at least two of the parts of each of these canonical forms, and so Theorem 9.1 applies and they cannot be referentially synonymous with $\mathsf{Ty}_2$ terms. 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. ⊣

**9.3. The new meanings in $\mathsf{L}^{\lambda}_{\mathbf{ar}}$ and logical form.** It can be argued that the "new meanings" of $\mathsf{L}^{\lambda}_{\rm ar}(K)$ which cannot be expressed by $\mathsf{Ty}_2$ terms are not mere curiosities. Consider the following two, related sentences, assuming for simplicity that "stumbled" and "fell" are denoted by constants:

(46a) John stumbled and he fell (coindexing)

(46b) John stumbled and fell (coordination)

Their rendering requires coindexing and coordination as indicated, and if we perform these operations using abstraction in the most natural way, 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 counterintuitive, especially as the renderings have the same logical form, which is not true of the English sentences: (46a) is a conjunction, while (46b) is a predication. If we do the coindexing and the coordination using the recursion construct (as in Sections 4.8 and 4.11 again), we get instead

(47a) John stumbled and he fell

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

(47b) John stumbled and fell

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

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

It is clear from the indicated canonical forms of these two terms that they are not synonymous and they render correctly (46a) as a conjunction and (46b) as a predication. In fact, easily,

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

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

so that the explicit rendering captures coordination correctly, in this example, while it misses on the coindexing: the formal term in (47a) 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 (46a) and (46b) 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.

**9.4. The symmetry of identity statements.** Let us also keep the promise made in Section 4.9, 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.,

$$(48)\ \mathrm{ES}_{\mathrm{Mont}}(\lambda(u)\mathrm{MS}_{\mathrm{Mont}}(\lambda(v)(u=v))) \not\approx \mathrm{MS}_{\mathrm{Mont}}(\lambda(u)\mathrm{ES}_{\mathrm{Mont}}(\lambda(v)(u=v)))$$

We assume that

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

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

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 "$\text{the}_{\text{Mont}}$" acts on relations and produces quantifiers, i.e.,

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

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

Proof of (48). Assume the opposite, and also, for simplicity, at first, that $\mathsf{first}$ and $\mathsf{last}$ are constants. Easily,

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

The subsequent reduction of these terms to canonical form will not affect their heads and the assignments to $\dot{h}$ and $\dot{p}$ which are already explicit and irreducible, and so by Theorem 5.3.1 we will have

$$\begin{aligned}
&\text{ES}_{\text{Mont}}(\lambda(u)\text{MS}_{\text{Mont}}(\lambda(v)(u = v))) \\
&\qquad \Rightarrow_{\text{cf}} \text{the}_{\text{Mont}}(\dot{p}_i) \text{ where } \{\dot{p}_i := \mathsf{first}(\dot{p}_j), \dot{p}_1 := A_1, \dots, \dot{p}_s := A_s\},
\end{aligned}$$

$$\begin{aligned}
&\text{MS}_{\text{Mont}}(\lambda(u)\text{ES}_{\text{Mont}}(\lambda(v)(u = v))) \\
&\qquad \Rightarrow \text{the}_{\text{Mont}}(\dot{p}_k) \text{ where } \{\dot{p}_k := \mathsf{last}(\dot{p}_l), \dot{p}_1 := B_1, \dots, \dot{p}_s := B_s\}.
\end{aligned}$$

If these two terms are to be referentially synonymous, then by the first condition on synonymy (RS1), we must have

$$\dot{p}_i \equiv \dot{p}_k, \text{ and hence } \dot{p}_j \equiv \dot{p}_l;$$

which then gives the absurd

$$\mathfrak{M}_0 \models \mathsf{first}(\dot{p}_j) = \mathsf{last}(\dot{p}_j).$$

The argument is just a bit more tedious if $\mathsf{first}$ and $\mathsf{last}$ are complex terms. ⊣

**10. Local (situated) meanings.** 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 and my correct identification of "John"" and "Mary" 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 *local* (situated) *meanings* of Carnap intensions and individual concepts.

In this section we consider how local meanings are represented and computed in $\mathsf{L}^{\lambda}_{\mathrm{ar}}$, preparatory to the modeling of propositional attitudes that we will introduce further down. The key idea is that the *global meaning* of a Carnap intension $A : \tilde{\mathsf{t}}$ is *a matter of language* only, while the local meaning of $A$ at a specific state $a$ is also *a matter of fact*.

**10.1. Utterances and local synonymy.** An *utterance*[32] 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 $\mathrm{g}(\overline{a})$ of every valuation 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.

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

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

is very simple 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 shape and 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* $a$ and similarly for *local logical synonymy*:

$$A \approx_a B \iff A(\overline{a}) \approx B(\overline{a}), \quad A \approx_{\ell,a} B \iff A(\overline{a}) \approx_{\ell} B(\overline{a}).$$

Notice that the axioms and rules of Table 4 hold also for these local versions of synonymy, because utterances $A(\overline{a})$ are just terms, albeit of a special form. The axiom system is also complete for these local synonymies, but there is no surprise in this; because (with the usual conventions about canonical forms and

[32] 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.

disregarding the trivial conditions on occurrences of free variables),

$$
\begin{aligned}
&(\text{for all } a)(A \approx_a B)\\
&\quad\iff (\text{for all } a)\Big(A_0(\overline{a}) \text{ where } \{\dot{p}_1 := A_1, \dots, \dot{p}_n := A_n\}\\
&\qquad\qquad \approx B_0(\overline{a}) \text{ where } \{\dot{p}_1 := B_1, \dots, \dot{p}_n := B_n\}\Big)\\
&\quad\iff (\text{for all } a)\Big(\mathfrak{M}_0 \models A_0(\overline{a}) = B_0(\overline{a})\Big) \,\&\, \bigwedge_{i=1,\dots,n} \mathfrak{M}_0 \models A_i = B_i\\
&\qquad\iff \mathfrak{M}_0 \models A_0 = B_0 \,\&\, \bigwedge_{i=1,\dots,n} \mathfrak{M}_0 \models A_i = B_i \iff A \approx B
\end{aligned}
$$

and similarly for logical synonymy.

With these notions in place, we can now analyze two standard, well-known puzzles in the philosophy of language. First a much simplified, monolingual version of the *Pierre puzzle* in Kripke [1979].

**10.2. Los Angeles and LA.** *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:*

(49a) 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*:

(49b) 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

$$\mathfrak{M}_0 \models \mathsf{Los\ Angeles} = \mathsf{LA}.$$

This is reasonable, as both abbreviations of the full name of Los Angeles are well established,[33] and it implies that

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

since Los Angeles and LA are explicit and irreducible. The Compositionality Theorem 5.3.4 now yields

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

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.

[33] 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,

(53) 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 (49a) and (49b), in other words, that Petros is not a language speaker—he is incoherent.

**10.3. 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 (53), 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 (50), which implies the synonymy (51). Thus, what we need to assume of a language speaker is that (at a minimum) *he knows all true identities*

(54) $\mathfrak{M}_0 \models \mathsf{a} = \mathsf{b}$

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 five years of living in Los Angeles, Moschovakis still makes no claim that he can can "grasp it" (whatever that means), but he certainly knows that

$\mathfrak{M}_0 \models \mathsf{Los\ Angeles} = \mathsf{LA}$,

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

We now turn to the second puzzle, which was introduced by Salmon and Soames in the introduction to Church [1982] and which is worth quoting verbatim.[34]

**10.4. 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*

(55a) He is not Scott

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

(55b) Scott is not Scott.

Now the puzzle comes from the circumstance that

(55c) $\mathrm{He}(a) = \mathrm{Scott}(a)$

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)

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

and that would make King George guilty of incoherence. But (56) is not true:

(57) $\Big(\text{He is Scott}\Big)(\overline{a}) \xrightarrow{\text{render}} \Big(\mathsf{He} = \mathsf{Scott}\Big)(\overline{a})$
$\Rightarrow_{\mathrm{cf}} (h = s)(\overline{a})$ where $\{h := \mathsf{He}, s := \mathsf{Scott}\}$
$\approx h(\overline{a}) = s(\overline{a})$ where $\{h := \mathsf{He}, s := \mathsf{Scott}\}$,

[34] Salmon and Soames started from a related puzzle of Church [1982], which is also about belief but employs variables rather than descriptions or demonstratives.

$$(58)\quad \Big(\text{Scott is Scott}\Big)(\overline{a}) \xrightarrow{\text{render}} \Big(\mathsf{Scott} = \mathsf{Scott}\Big)(\overline{a})$$
$$\Rightarrow_{\text{cf}} (s' = s)(\overline{a}) \text{ where } \{s' := \mathsf{Scott}, s := \mathsf{Scott}\}$$
$$\approx s'(\overline{a}) = s(\overline{a}) \text{ where } \{s' := \mathsf{Scott}, s := \mathsf{Scott}\},$$

and by the Referential Synonymy Theorem 5.3.1

$$(59)\quad \Big(\mathsf{He} = \mathsf{Scott}\Big)(\overline{a}) \not\approx \Big(\mathsf{Scott} = \mathsf{Scott}\Big)(\overline{a}),$$

simply because

$$\mathfrak{M}_0 \models \mathsf{He} \neq \mathsf{Scott}.$$

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

**10.5. 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.*[35]

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 ≠ Scott, which saves the King. We have already discussed in Section 10.1 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 $\mathsf{c}$.

**10.6. 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

$$(60a)\quad \mathsf{He}(\overline{a}) = \mathsf{Scott}(\overline{a}),$$

which seems to mean exactly the same as

$$(60b)\quad \mathsf{Scott}(\overline{a}) = \mathsf{Scott}(\overline{a}).$$

Well, if the King had actually denied (60a), then, indeed, we would have had a puzzle, because by compositionality and (55c),

$$(60c)\quad \mathsf{He}(\overline{a}) = \mathsf{Scott}(\overline{a}) \approx \mathsf{Scott}(\overline{a}) = \mathsf{Scott}(\overline{a}).$$

[35] 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.

But the King did not deny (60a), and, indeed, *he could not have denied* (60a) *because* (60a) *is not an utterance.*[36] 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.

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

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

If we replace the state parameter $\overline{a}$ by the constant $\mathsf{book\text{-}signing}$ in the terms of (60c), we get

(61a) $\mathsf{He}(\mathsf{book\text{-}signing})(\overline{a}) = \mathsf{Scott}(\mathsf{book\text{-}signing})(\overline{a})$,

(61b) $\mathsf{Scott}(\mathsf{book\text{-}signing})(\overline{a}) = \mathsf{Scott}(\mathsf{book\text{-}signing})(\overline{a})$,

and the King can try to deny the first while asserting the second. But the constant $\mathsf{He}$ is a part of (61a) and not (denotationally) equal to any part of (61b), and so these two utterances are not synonymous[37] 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.

---

[36] It may be argued that (by the Gallin interpretation), (60a) is exactly what the King denies when his utterance "He is not Scott" is rendered in $\mathsf{LIL}$ and $a$ is the current state. This is discussed in Kalyvianaki and Moschovakis [2008]. The claim there is that as renderings of utterances in $\mathsf{LIL}$ are naturally understood, they cannot serve as belief carriers—although they express a robust notion of *information* or *factual content*, related to but different from local meaning. Kalyvianaki [2007] gives an extensive treatment of factual content which, unfortunately, we cannot discuss here.

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

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

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

**11. Propositional attitudes.** We now extend the theory of referential intensions to allow in the lexicon $K$ names for *propositional attitudes* such as

say that ... , claim that ... , know that ... , believe that ... .

The problem we must face is that the truth and the meaning of

Nixon claimed that he is not a crook

$$\xrightarrow{\text{render}} \mathsf{Claimed}(\dot{n}, \mathsf{not}(\mathsf{crook}(\dot{n}))) \text{ where } \{\dot{n} := \mathsf{Nixon}\}$$

depend not only on the truth value of $\mathsf{crook}(\mathsf{Nixon})$ but also its meaning; otherwise, if Nixon spoke the truth, then he claimed every truth on that fateful day, which is unlikely. This means that $\mathsf{Claimed}$ must be interpreted by a function which takes a local referential intension for its second argument; and the method for attaching meanings to such terms exploits the fact that referential intensions are (basically) tuples of functions in our universe $\mathfrak{M}_0$, and so $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ can talk about them.

Our basic assumption about the claiming relation which must be reflected in our modeling of it is that for any two, proper Carnap intensions, $A, B : \tilde{\mathsf{t}}$ and any state $a$,

(62) if $A \approx_a B$ and $x$ claims $A$ in $a$, then $x$ claims $B$ in $a$.

The task is somewhat complicated by the need to allow free variables in $A, B$ as in $\mathsf{not}(\mathsf{crook}(\dot{n}))$ above, to account for coindexing that is resolved outside the claiming relation and other instances of *quantifying in*.

**11.1. Formal attitudinal application.** The new *attitudinal constants* are of type[38]

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

and they can be used to construct $\mathsf{L}^{\lambda}_{\mathrm{ar}}$-terms by the following rule:

*If* $\mathsf{C}$ *is an attitudinal constant,* $A : \tilde{\mathsf{e}}$ *is an arbitrary term and* $B : \tilde{\mathsf{t}}$ *is proper, then*

$$\mathsf{C}(A, B) : \tilde{\mathsf{t}}.$$

We insist that the second argument $B$ be a proper Carnap intension, because it must have meaning for the attitudinal application to make sense. Notice that these attitudinal operations are used "syncategorematically", i.e., $\mathsf{Claims}$ by itself is not a term. But they can be nested and be combined with all the other constructs of $\mathsf{L}^{\lambda}_{\mathrm{ar}}$ to yield terms like

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

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

A term $A$ is *denotational* if no attitudinal constant occurs in $A$.

[38] There are interesting unary propositional attitudes, like "it is commonly known that ... " as well as ternary ones, like "$x$ told $y$ that ... ". Everything we do in this section extends to them trivially, and it simplifies matters to deal only with the binary ones.

Before we outline the general construction of canonical forms and the definition of referential intensions of attitudinal terms, we illustrate the idea with some simple examples, leaving the rigorous justification of the computations for later.

**11.2. George claims that John is a crook.** By the reduction calculus as we know it, and assuming that it applies to denotational terms even when they are arguments of an attitudinal constant:

$$\mathsf{Claims}(\mathsf{George}, \mathsf{crook}(\mathsf{John})) \Rightarrow \mathsf{Claims}(\mathsf{George}, \mathsf{crook}(j) \text{ where } \{j := \mathsf{John}\}).$$

The denotation of this depends on the referential intension of

$$CJ :\equiv \mathsf{crook}(j) \text{ where } \{j := \mathsf{John}\},$$

so we compute it as in Section 7:

$$\begin{aligned} \mathfrak{s} = \mathrm{shape}(CJ) &= (j, \emptyset, \langle j \rangle, \emptyset, \emptyset) \\ \mathrm{system}(CJ) &= (\alpha_0, \alpha_1), \\ \mathrm{int}(CJ) &= (\mathfrak{s}, \mathrm{system}(CJ)) \end{aligned}$$

where

$$\alpha_0(\mathrm{g}, \jmath) = \mathrm{crook}(\jmath), \quad \alpha_1(\mathrm{g}) = \mathrm{John}.$$

These functions $\alpha_0, \alpha_1$ do not depend on the valuation g, because we have exhibited the variables on which each of them depends, getting them from the shape. So the "reduced functions"

$$\alpha_0'(\jmath) = \mathrm{crook}(\jmath), \quad \alpha_1'(\,) = \mathrm{John}$$

determine system($CJ$), and they are objects in our universe. They suggest the next, crucial move in the analysis:

$$\begin{aligned} &\mathsf{Claims}(\mathsf{George}, CJ) \Rightarrow \mathsf{Claims}^{\mathfrak{S}}(\mathsf{George}, \mathsf{fint}(\mathsf{crook}(\mathsf{John})) \\ &\quad \equiv \mathsf{Claims}^{\mathfrak{S}}(\mathsf{George}, \mathsf{fint}(\mathsf{crook}(j) \text{ where } \{j := \mathsf{John}\})) \\ &\qquad\qquad \equiv \mathsf{Claims}^{\mathfrak{S}}(\mathsf{George}, \lambda(\jmath)\mathsf{crook}(\jmath), \mathsf{John}) \end{aligned}$$

where $\mathsf{fint}(CJ)$ stands for the (formal) representation of int($CJ$) and $\mathsf{Claims}^{\mathfrak{S}}$ is a *new, denotational constant* which, in effect, computes the truth value of $A$ from the referential intension of $CJ$. (And we will define both of these in the sequel, shortly, but it should be clear how to do it from the discussion.)

**11.3. Declaration of love.** Suppose we have a constant Declares and compute as usual:

Peter declares that he loves John's sister

$$\begin{aligned} &\xrightarrow{\text{formalize}} \mathsf{Declares}(\mathsf{Peter}, \mathsf{loves}(\mathsf{he}, \mathsf{sister}(\mathsf{John}))) \\ &\xrightarrow{\text{coindex}} A \equiv \mathsf{Declares}(\dot{p}, \mathsf{loves}(\dot{p}, \mathsf{sister}(\mathsf{John}))) \text{ where } \{\dot{p} := \mathsf{Peter}\} \\ &\Rightarrow \mathsf{Declares}\Big(\dot{p}, \mathsf{loves}(\dot{p}, \dot{s}) \text{ where } \{\dot{s} := \mathsf{sister}(j), j := \mathsf{John}\}\Big) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{where } \{\dot{p} := \mathsf{Peter}\} \\ &\equiv \mathsf{Declares}(\dot{p}, L) \text{ where } \{\dot{p} := \mathsf{Peter}\} \end{aligned}$$

with the abbreviation

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

If the denotation of $\mathsf{Declares}(\dot{p}, L)$ depended only on the denotation of $L$, we would apply the ap rule next to get

$$\mathsf{Declares}(\dot{p}, L) \Rightarrow \mathsf{Declares}(\dot{p}, \dot{L}) \text{ where } \{\dot{L} := L\} \qquad \text{(False!)}$$

and then proceed as usual to get the canonical form of $A$. But this is clearly wrong. So we compute $\mathrm{int}(L)$ as in Section 7:

$$\begin{aligned} \mathfrak{s} = \mathrm{shape}(L) &= (\dot{s}, j, \langle \dot{p} \rangle, \langle \dot{s} \rangle, \emptyset, \langle j \rangle, \emptyset, \emptyset) \\ \mathrm{system}(L) &= (\alpha_0, \alpha_1, \alpha_2), \\ \mathrm{int}(L) &= (\mathfrak{s}, \mathrm{system}(L), \end{aligned}$$

where

$$\alpha_0(\mathrm{g}, p, s) = \mathrm{loves}(p, s), \quad \alpha_1(\mathrm{g}, j) = \mathrm{sister}(j), \quad \alpha_2(\mathrm{g}) = \mathrm{John}.$$

As in the example above, these functions do not depend on the valuation g, and so the "reduced functions"

$$\alpha_0'(p, s) = \mathrm{loves}(p, s), \quad \alpha_1'(j) = \mathrm{sister}(j), \quad \alpha_2 = \mathrm{John}$$

determine $\mathrm{system}(L)$, and they are objects in our universe. They suggest the next, crucial move in the analysis:

(64) $$\mathsf{Declares}(\dot{p}, L) \Rightarrow \mathsf{Declares}^{\mathbf{5}}(\dot{p}, \dot{p}, \mathsf{fint}(L)) \equiv \mathsf{Declares}^{\mathbf{5}}(\dot{p}, \dot{p}, \lambda(p, s)\mathsf{loves}(p, s), \lambda(j)\mathsf{sister}(j), \mathsf{John})$$

where $\mathsf{Declares}^{\mathbf{5}}$ is a *new, denotational constant* which, in effect, computes the truth value of $A$ from the referential intension of $L$; the second occurrence of $\dot{p}$ signifies that $\dot{p}$ occurs freely in $L$. Leaving aside for a moment the precise interpretation of $\mathsf{Declares}^{\mathbf{5}}$, we have a formal reduction

(65) $$A \equiv \mathsf{Declares}(\dot{p}, \mathsf{loves}(\dot{p}, \mathsf{sister}(\mathsf{John}))) \text{ where } \{\dot{p} := \mathsf{Peter}\}$$
$$\Rightarrow \mathsf{Declares}^{\mathbf{5}}(\dot{p}, \dot{p}, \lambda(p, s)\mathsf{loves}(p, s), \lambda(j)\mathsf{sister}(j), \mathsf{John}) \text{ where } \{\dot{p} := \mathsf{Peter}\}$$

from which we can continue as usual to get the canonical form of $A$,

(66) $$A \Rightarrow_{\mathrm{cf}} \mathsf{Declares}^{\mathbf{5}}(\dot{p}, \dot{p}, \dot{l}, \dot{s}, j)$$
$$\text{where } \{\dot{p} := \mathsf{Peter}, \dot{l} := \lambda(p, s)\mathsf{loves}(p, s), \dot{s} := \lambda(j)\mathsf{sister}(j), j := \mathsf{John}\}$$
$$\approx_{\ell} \mathsf{Declares}^{\mathbf{5}}(\dot{p}, \dot{p}, \dot{l}, \dot{s}, j) \text{ where } \{\dot{p} := \mathsf{Peter}, \dot{l} := \mathsf{love}, \dot{s} = \mathsf{sister}, j := \mathsf{John}\}.$$

For this to give us the correct answer, we must define the interpretation $\mathsf{Declares}^{\mathbf{5}}$ of the new constant so that for any state $a$,

(67) $$\mathsf{Declares}^{\mathbf{5}}(\mathrm{Peter}, \mathrm{Peter}, \mathrm{love}, \mathrm{sister}, \mathrm{John})(a) = 1$$
$$\iff \text{in state } a, \text{ Peter declares that he loves John's sister,}$$

which, presumably, can be empirically verified. To see how to do this, we check first directly from the definitions, that for any term $B$ and state $a$,

$$
\begin{aligned}
(68)\quad B(\overline{a}) \approx L(\overline{a}) \iff\ & B \Rightarrow_{\mathrm{cf}} B_0 \text{ where } \{\dot{s} := B_1, j := B_2\} \\
& \text{for some } B_0, B_1, B_2, \text{ such that} \\
& \quad \mathrm{shape}(B_0 \text{ where } \{\dot{s} := B_1, j := B_2\}) = \mathrm{shape}(L) = \mathfrak{s} \\
& \quad \text{and } \mathfrak{M}_0 \models \lambda(p,s)(\mathsf{love}(p,s)(\overline{a})) = \lambda(p,s)(B_0\{\dot{p} := p, \dot{s} := s\}(\overline{a})) \\
& \quad\quad \text{and } \mathfrak{M}_0 \models \mathsf{sister} = \lambda(\jmath)B_1\{j := \jmath\} \text{ and } \mathfrak{M}_0 \models \mathsf{John} = B_2.
\end{aligned}
$$

So we set, for each state $a$,[39]

$$
\begin{aligned}
&\mathrm{Declares}^{\mathfrak{s}}(p,q,l,s,\jmath)(a) = 1 \\
&\quad \iff \text{there is an irreducible term } B \equiv B_0 \text{ where } \{\dot{s} := B_1, j := B_2\} \\
&\qquad \text{such that } \mathrm{shape}(B_0 \text{ where } \{\dot{s} := B_1, j := B_2\}) = \mathfrak{s} \\
&\qquad \text{and } \lambda(p,s)l(p,s,a) = \lambda(p,s)(\mathrm{den}(B_0)(\{\dot{p} := p, \dot{s} := s\})(a)) \\
&\qquad \text{and } s = \lambda(\jmath)\mathrm{den}(B_1)(\{j := \jmath\}) \text{ and } \jmath = \mathrm{den}(B_2) \\
&\qquad\qquad \text{and } p \text{ declares in state } a \text{ that } B.
\end{aligned}
$$

We set $\mathrm{Declares}^{\mathfrak{s}}(p,q,l,s,\jmath)(a) = 0$ if the condition on the right is not satisfied, and it is then immediate that the right-to-left direction of (67)

in state $a$, Peter declares that he loves John's sister
$$\Longrightarrow \mathrm{Declares}^{\mathfrak{s}}(\mathrm{Peter}, \mathrm{Peter}, \mathrm{love}, \mathrm{sister}, \mathrm{John})(a) = 1$$

holds, by taking $B = L$. The left-to-right direction also holds, because the hypothesis

$$\mathrm{Declares}^{\mathfrak{s}}(\mathrm{Peter}, \mathrm{Peter}, \mathrm{love}, \mathrm{sister}, \mathrm{John})(a) = 1$$

supplies us with a term $B \equiv B_0$ where $\{\dot{s} := B_1, j := B_2\}$ such that Peter declares $B$ in state $a$ and $B(\overline{a}) \approx L(\overline{a})$ by (68); and so by (62), if he is coherent, this amounts to a declaration of his love for John's sister too.

For the general construction below, we will need one more simple but important syntactic construct.

**11.4. The $\lambda^{\mathrm{r}}$ operator.** There was an important move in (64) that we did not discuss, when the recursion variables $\dot{p}, \dot{s}$ on $\mathsf{loves}(\dot{p}, \dot{s})$ on the left side (within $L$) were changed to pure variables in the $\lambda$-abstraction $\lambda(p,s)\mathsf{loves}(p,s)$ on the right. It was a necessary move, as we have not allowed $\lambda$-abstraction on recursion variables, and for good reason: the separate uses of the two kinds of variables are crucial in formulating correctly the reduction calculus. From the denotational point of view, however, there is nothing wrong with recursion variable $\lambda$-abstraction, since these variables range over exactly the same sets as the pure

[39] Note that $q$ is not used in this definition, which implicitly assumes that it is equal to $p$, which in this case means that the person making the claim is the same as that about whom the claim is made.

variables. It is convenient now to introduce this construct, as an abbreviation. We set

$$(69)\quad \lambda^{\mathrm{r}}(x)(A) :\equiv \begin{cases} \lambda(x)(A), & \text{if } x \text{ is a pure variable,} \\ \lambda(x')(A\{x :\equiv x'\}), & \text{if } x \text{ is a recursion variable,} \end{cases}$$

where, in the second case, $x'$ is a fresh, pure variable of the same type as $x$.

Notice that if $A$ is immediate or irreducible, then $\lambda^{\mathrm{r}}(x)(A)$ is also immediate or irreducible accordingly, directly from the definition of immediate terms in Section 6.3 and the characterization of irreducible terms in Section 6.7. This is an important property of the $\lambda^{\mathrm{r}}$ construct.

**11.5. Outline of the general construction.** Suppose now that

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

is an arbitrary attitudinal constant with which we can form terms $\mathsf{C}(A, B)$ as explained in Section 11.1. As in the special cases above, we will associate with it a family $\mathsf{C}^{\mathfrak{s}}$ of denotational constants, one for each "abstract" shape $\mathfrak{s}$, so that (roughly)

$\mathsf{C}^{\mathfrak{s}}$ will be used to interpret $\mathsf{C}(A, B)$ if $\mathrm{shape}(B) = \mathfrak{s}$.

Specifically, if

$$\mathfrak{s} = (\dot{p}_1, \dots, \dot{p}_n, \vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0, \dots, \vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)$$

as in (AR1) and (AR2) of (44) with $\dot{p}_i : \sigma_i$ for $i = 1, \dots, n$, then

$$\mathsf{C}^{\mathfrak{s}} : \tilde{\mathsf{e}} \times \mathrm{type}(\vec{\mathbf{f}}) \times \sigma_0^* \times \sigma_1^* \times \cdots \times \sigma_n^* \to \tilde{\mathsf{t}}$$

where $\vec{\mathbf{f}} = \vec{\mathbf{f}}_0 \cup \cdots \cup \vec{\mathbf{f}}_n$ is the sequence variables which occur free in $B$ (in some standard enumeration without repetitions), $\mathrm{type}(\vec{\mathbf{f}})$ is the sequence of their types (and it is simply omitted if $\vec{\mathbf{f}} = \emptyset$), $\sigma_0 \equiv \tilde{\mathsf{t}}$, and

$$\sigma_i^* :\equiv \mathrm{type}(\vec{\mathbf{f}}_i) \times \mathrm{type}(\vec{\mathbf{r}}_i) \to \sigma_i,$$

with the same understanding of $\mathrm{type}(\vec{\mathbf{r}}_i)$. The interpretations $C^{\mathfrak{s}}$ of these constants are assumed to be given "empirically", from our understanding of the propositional attitude $C$, as we specify below.

The reduction calculus is exactly as before, except for one restriction and one addition:

**Restriction**: The ap rule

$$A(B) \Rightarrow A(\dot{b}) \text{ where } \{\dot{b} := B\}$$

can only be applied if $B$ is proper and *denotational*, i.e., no attitudinal constant occurs in $B$.

**11.5.1. The attitudinal application rule.** Suppose

$$B \equiv B_0 \text{ where } \{\dot{p} := B_1, \dots \dot{p}_n := B_n\} : \tilde{\mathsf{t}}$$

is denotational, proper and irreducible,

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

$$\mathsf{C}(A,B) \Rightarrow \mathsf{C}^{\mathfrak{s}}(A,\vec{\mathbf{f}},\mathsf{fint}(B)) \qquad \text{(attap)}$$

$B \equiv B_0$ where $\{\dot{p}_1 := B_1, \dots, \dot{p}_n := B_n\}$ is proper, irreducible

$\mathsf{fint}(B) \equiv (\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0,\vec{\mathbf{r}}_0)(B_0), \dots, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n,\vec{\mathbf{r}}_n)(B_n))$

TABLE 9. The reduction calculus: the attitudinal application rule.

in the notation of Section 1, and $\mathsf{C}$ is an attitudinal constant. Let

(70) $\mathsf{fint}(B) \equiv (\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0,\vec{\mathbf{r}}_0)(B_0), \dots, \lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n,\vec{\mathbf{r}}_n)(B_n))$

be the tuple of $\lambda$-terms which formally defines the (reduced) intension of $B$ as above, and set

$$\mathsf{C}(A,B) \Rightarrow \mathsf{C}^{\mathfrak{s}}(A,\vec{\mathbf{f}},\mathsf{fint}(B)) \qquad \text{(attap)}$$

Notice that the terms $\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_i,\vec{\mathbf{r}}_i)(B_i)$ are irreducible, and the two sides of (attap) have the same free variables—this is the point of including the list $\vec{\mathbf{f}}$ on the right.

For the example in Section 11.3 (with "Declares"), this gives exactly what we "guessed" there,

$$\mathsf{Declares}(\dot{p},\mathsf{loves}(\dot{p},\dot{s}) \text{ where } \{\dot{s} := \mathsf{sister}(j), j := \mathsf{John}\}) \Rightarrow \mathsf{Declares}^{\mathfrak{s}}(\dot{p},\dot{p},\lambda(p,s)\mathsf{loves}(p,s),\lambda(j)\mathsf{sister}(j),\mathsf{John})$$

where $\mathfrak{s} = \mathrm{shape}(\mathsf{loves}(\dot{p},\dot{s}) \text{ where } \{\dot{s} := \mathsf{sister}(j), j := \mathsf{John}\})$.

It is important here that from the list $\vec{\mathbf{f}}$ of free variables of $B$ and the shape $\mathfrak{s}$ we can compute the lists $\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i$ of the variables which occur free in each $B_i$ and which are needed to construct the terms in the tuple $\mathsf{fint}(B)$.

**11.6. Denotational soundness.** We check that the attap rule preserves denotations, with the correct definition of the denotational constants $C^{\mathfrak{s}}$ and the assumption that the propositional attitude satisfies the basic principle (62). In the notation we used to state the rule and using $\mathsf{Claims}$ to keep the definition concrete, set:

$\mathrm{Claims}^{\mathfrak{s}}(x,\vec{\mathbf{f}},b_0,b_1,\dots b_n)(a) = 1$

$\iff$ there is an irreducible term

$B \equiv B_0$ where $\{\dot{p}_1 := B_1, \dots, \dot{p}_n := B_n\} : \tilde{\mathsf{t}}$ with $\mathrm{shape}(B) = \mathfrak{s}$, such that

$\lambda(\vec{\mathbf{f}}_0,\vec{\mathbf{r}}_0)(b_0(\vec{\mathbf{f}}_0,\vec{\mathbf{r}}_0,a)) = \mathrm{den}((\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_0,\vec{\mathbf{r}}_0)(B_0(\vec{\mathbf{f}}_0,\vec{\mathbf{r}}_0)(\overline{a}))),$

$b_1 = \mathrm{den}(\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_1,\vec{\mathbf{r}}_1)B_1) \dots, b_n = \mathrm{den}(\lambda^{\mathrm{r}}(\vec{\mathbf{f}}_n,\vec{\mathbf{r}}_n)B_n)$

and in state $a$, $x$ claims $B$.

If the condition on the right does not hold, we set

$\mathrm{Claims}^{\mathfrak{s}}(x,\vec{\mathbf{f}},b_0,b_1,\dots b_n)(a) = 0.$

Notice that the term $B$ in this definition may have free variables, as

$L \equiv \mathsf{loves}(\dot{p},\dot{s})$ where $\{\dot{s} := \mathsf{sister}(j), j := \mathsf{John}\}$

does in Section 11.3 above, and so we must interpret correctly claims of Carnap intensions with free variables. We understand that $\dot{p}$ in $L$ refers directly and immediately to some individual concept (presumably a person), who claims in state $a$ that the local meaning $B$ is true—as he understands $B$ in state $a$. It is important to allow this, so we can interpret claims about "him" or de re claims about "the tallest man in the room".

(1) *If $x$ claims an irreducible Carnap intension*

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

*in state $a$ and* $\text{shape}(D) = \mathfrak{s}$*, then*

$$\text{Claims}^{\mathfrak{s}}(x, \vec{\mathbf{f}}, \lambda^{\text{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)D_0, \dots, \lambda^{\text{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)D_n)(a) = 1.$$

This is immediate, taking $B \equiv D$ in the definition of $\text{Claims}^{\mathfrak{s}}$.

(2) *If*

$$\text{Claims}^{\mathfrak{s}}(x, \vec{\mathbf{f}}, \lambda^{\text{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)D_0, \dots, \lambda^{\text{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)D_n)(a) = 1,$$

*then $x$ claims $D$ in state $a$.*

The hypothesis gives us an irreducible term $B$ with $\text{shape}(B) = \mathfrak{s} = \text{shape}(D)$ such that $x$ claims $B$ in state $a$, and

$$\text{(71a)} \quad \mathfrak{M}_0 \models \lambda^{\text{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)(B_0(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)(\overline{a})) = \lambda^{\text{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)(D_0(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)(\overline{a})),$$

$$\text{(71b)} \quad \mathfrak{M}_0 \models \lambda^{\text{r}}(\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i)B_i = \lambda^{\text{r}}(\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i)D_i \text{ for } i = 1, \dots, n.$$

In addition, for every $i = 0, \dots, n$,

$$\text{the terms } \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i)B_i\Big)(\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i) \text{ and } \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i)D_i\Big)(\vec{\mathbf{f}}_i, \vec{\mathbf{r}}_i) \text{ are irreducible.}$$

Put now

$$B' :\equiv \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)B_0\Big)(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)$$
$$\text{where } \{\dot{p}_1 := \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_1, \vec{\mathbf{r}}_1)B_1\Big)(\vec{\mathbf{f}}_1, \vec{\mathbf{r}}_1), \dots, \dot{p}_n := \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)B_n\Big)(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)\},$$

and similarly

$$D' :\equiv \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)D_0\Big)(\vec{\mathbf{f}}_0, \vec{\mathbf{r}}_0)$$
$$\text{where } \{\dot{p}_1 := \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_1, \vec{\mathbf{r}}_1)D_1\Big)(\vec{\mathbf{f}}_1, \vec{\mathbf{r}}_1), \dots, \dot{p}_n := \Big(\lambda^{\text{r}}(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)D_n\Big)(\vec{\mathbf{f}}_n, \vec{\mathbf{r}}_n)\},$$

Now $B'$ and $D'$ are both irreducible and the parts of $B'(\overline{a})$ and $D'(\overline{a})$ have the same denotations by (71a), (71b), so

$$B'(\overline{a}) \approx D'(\overline{a});$$

and by the same reasoning,

$$B(\overline{a}) \approx B'(\overline{a}) \text{ and } D(\overline{a}) \approx D'(\overline{a});$$

and since $x$ claims $B$ in state $a$, he must also claim the locally synonymous $D$ in state $a$ by (62), which is what we needed to prove.

**11.7. "John claims Mary loves him but she denies it".** Formalizing and then doing the familiar coindexing operations as usual, we get

John claims Mary loves him but she denies it

$$\xrightarrow{\text{formalize}} A \equiv \mathsf{Claims}(\mathsf{John}, \mathsf{loves}(\mathsf{Mary}, \mathsf{him}))\ \mathsf{but}\ \mathsf{Denies}(\mathsf{she}, \mathsf{it})$$

$$\xrightarrow{\text{coindex}} \mathsf{Claims}(j, \mathsf{loves}(\dot{m}, j))\ \mathsf{but}\ \mathsf{Denies}(\dot{m}, \mathsf{it})\ \mathsf{where}\ \{j := \mathsf{John}, \dot{m} := \mathsf{Mary}\}.$$

There is another coindexing that is needed, however, since it clearly refers of $\mathsf{loves}(\dot{m}, j)$—which means *to the meaning of* this sentence; so we would like to justify continuing with

$$\xrightarrow{\text{coindex}} \mathsf{Claims}(j, \dot{l})\ \mathsf{but}\ \mathsf{Denies}(\dot{m}, \dot{l})$$
$$\mathsf{where}\ \{j := \mathsf{John}, \dot{m} := \mathsf{Mary}, \dot{l} = \mathsf{fint}(\mathsf{loves}(\dot{m}, j))\}$$

It is clear that the shape of $\mathsf{loves}(\dot{m}, j)$ will play a role, so set

$$\mathfrak{s} := \mathrm{shape}(\mathsf{loves}(\dot{m}, j)) = (\dot{m}, j, \langle \dot{m}, j\rangle, \emptyset),$$

and then proceed formally:

$$\Rightarrow \mathsf{Claims}^{\mathfrak{s}}(j, \dot{m}, j, \dot{l})\ \mathsf{but}\ \mathsf{Denies}^{\mathfrak{s}}(\dot{m}, \dot{m}, j, \dot{l})$$
$$\mathsf{where}\ \{j := \mathsf{John}, \dot{m} := \mathsf{Mary}, \dot{l} := \lambda(m, \jmath)\mathsf{loves}(m, \jmath)\}$$
$$\approx_\ell \mathsf{Claims}^{\mathfrak{s}}(j, \dot{m}, j, \dot{l})\ \mathsf{but}\ \mathsf{Denies}^{\mathfrak{s}}(\dot{m}, \dot{m}, j, \dot{l})$$
$$\mathsf{where}\ \{j := \mathsf{John}, \dot{m} := \mathsf{Mary}, \dot{l} := \mathsf{love}\}.$$

The result looks right, and it is, if we recall how the (denotational) relations $\mathsf{Claims}^{\mathfrak{s}}, \mathsf{Denies}^{\mathfrak{s}}$ are interpreted: it says that

$j$ claims that $\dot{l}(\dot{m}, j)$ but $\dot{m}$ denies that $\dot{l}(\dot{m}, j)$
where $j$ is John, $\dot{m}$ is Mary, and $\dot{l}$ is love.

The example is quite easy because the sentence claimed by John and denied by Mary is rendered by an explicit, irreducible term and so has a very simple formal intension, with just one part, i.e.,

$$\mathsf{fint}(\mathsf{loves}(\dot{m}, j)) = \lambda(m, \jmath)\mathsf{loves}(m, \jmath)).$$

**11.8. The complete reduction calculus.** It should be clear that the addition of the attap rule extends the results in the preceding sections to the full language, with arbitrary attitudinal constants: these are eliminated, one-by-one, starting "from the inside" where the attitudinal constants are applied to terms with only denotational primitives, until we reach a canonical form in which only denotational constants occur, from which we read off the relevant referential intensions. The Referential Synonymy Theorem 5.3.1 holds by the same argument, and so do the rules for Logical and Referential Synonymy in Theorem 5.3.3, including the compositionality property. There is no compositionality for denotations in the full system, of course, and there should not be.

**11.9. "I believe everything that Sarah Palin says".** We have assumed an "empirical" understanding of attitudinal constants because that is the only possible interpretation that we can envision for *Claims that ...* , *Declares that ...* , etc., and the general, logical theory we are constructing should certainly cover these. The treatment is general enough to incorporate theoretical accounts of specific propositional attitudes like *common* or *potential knowledge*, *belief*, etc., typically given in modal terms. It is limited, however, in that we cannot express sentences like that in the heading: this is because of the basic, Fregean assumption we made, that the truth and meaning of a propositional attitude are functions of the local meaning of its argument, and the collection of local referential intensions do not form a set in our universe—or any structure into which the $\lambda$-calculus can be interpreted. Whether this is an unavoidable limitation of a Fregean approach to the theory of meaning or suggests a superiority of the modal treatments of propositional attitudes is a matter for further investigation.

**12. What is missing.** A lot, of course. We mention here only two important topics which should be treated in a logical theory of meaning and which we left out completely, because of lack of space in these notes and time in the lectures.

**12.1. Factual content.** This is the common, local semantic value of *He is Scott* and *Scott is Scott* in a state $a$ in which, in fact, "He" refers to Scott. It is different from the distinct local meanings of these two sentences, cf. Section 10.4 and Footnote 36. It is an important notion which has been analyzed in the context of referential intension theory in Kalyvianaki [2007].

**12.2. Approximate synonymy.** One can plausibly claim that no completely faithful translation into English can be given for some Greek phrases, simply because some words in the Greek lexicon do not have exact English counterparts. Still, we get by quite well with "approximate translations", and a theoretical account that justifies the practice should be a part of a logical theory of meaning. It, too, is left for another day.

REFERENCES

Alonzo Church [1982], *A remark conerning Quine's paradox about modality*, Spanish version in ***Analisis Filosófico***, pp. 25–32, reprinted in English in Salmon and Soames [1988].

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 University 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 [1879], ***Begriffsschrift. Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens (Halle)***, Translated by Stefan Bauer-Mengelberg in Van

Heijenoort [1967].

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

Harvey Friedman [1974], *Equality between functionals*, ***Logic Colloquium, Symposium on Logic held in Boston 1972-73*** (A. Dold and B. Eckman, editors), Lecture notes in mathematics, no. 453, Springer-Verlag, Berlin - Heidelberg - New York, pp. 22–37.

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 [2007], *Algorithmic natural language semantics: A study of Locality in the Theory of Referential Intensions*, ***Ph.D. thesis***, Graduate Program in Logic, Algorithms and Computation, University of Athens.

Eleni Kalyvianaki and Yiannis N. Moschovakis [2008], *Two aspects of situated meaning*, ***Logics for linguistic structures***, Mouton de Gruyter, Berlin, New York, pp. 57–86.

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].

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 [1989], *The formal language of recursion*, ***The Journal of Symbolic Logic***, vol. 54, pp. 1216–1252.

Yiannis N. Moschovakis [1994], *Sense and denotation as algorithm and value*, ***Logic colloquium '90*** (J. Väänänen and J. Oikkonen, editors), Lecture Notes in Logic, vol. 2, Association for Symbolic Logic, pp. 210–249, a corrected and expanded proof of the main theorem is posted in www.math.ucla.edu/∼ynm.

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.

Yiannis N. Moschovakis [2006], *A logical calculus of meaning and synonymy*, ***Linguistics and Philosophy***, vol. 29, pp. 27–89.

Jamal Ouhalla [1994], ***Introducing transformational grammar***, Arnold and Oxford University Press.

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.

Scott Soames [1989], *Presupposition*, ***Handbook of philosophical logic*** (D. Gabbay and F. Guenthner, editors), vol. IV, Reidel, pp. 553 – 616.

William Tait [1967], *Interpretations of functionals of finite type*, ***The Journal of Symbolic Logic***, vol. 32, pp. 198 – 212.

Pavel Tichý [1969], *Intensions in terms of Turing Machines*, ***Studia Logica: an international journal for symbolic logic***, vol. 24, pp. 7–25.

Jean Van Heijenoort (editor) [1967], ***From Frege to Gödel, a source book in mathematical logic, 1879 – 1931***, Harvard University Press, Cambridge, Massachusetts, London, England.

T. Zimmermann [1989], *Intensional logic and two–sorted type theory*, ***The Journal of Symbolic Logic***, vol. 54, pp. 65–77.

SEMINAR FÜR SPRACHWISSENSCHAFT UNIVERSITÄT TÜBINGEN
*E-mail*: friedrich.hamm@uni-tuebingen.de

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