Logic Colloquium '90
ASL meeting, Helsinki

SENSE AND DENOTATION AS ALGORITHM AND VALUE

Yiannis N. Moschovakis[1,2]

§1. Introduction.

In his classic 1892 paper *On sense and denotation* [12], Frege first contends that in addition to their **denotation** (*reference*, *Bedeutung*), proper names also have a **sense** (*Sinn*) "wherein the mode of presentation [of the denotation] is contained." Here proper names include common nouns like "the earth" or "Odysseus" and descriptive phrases like "the point of intersection of lines L_1 and L_2" which are expected by their grammatical form to name some object. By the second fundamental doctrine introduced by Frege in the same paper, they also include declarative sentences: "[a simple, assertoric sentence is] to be regarded as a proper name, and its denotation, if it has one, is either the True or the False." Thus every sentence *denotes* (or *refers to*) *its truth value* and *expresses its sense*, which is "a mode of presentation" of its truth value: this is all there is to the meaning of a sentence as far as logic is concerned. Finally, Frege claims that although sense and denotation are related (the first determines the second), they obey separate *principles of compositionality*, so that the truth value of a complex sentence ϕ is determined solely by the denotations of its constituent parts (terms and sentences), whatever the senses of these constituents parts may be. This is the basic principle which has made possible the development of sense-independent (two-valued, classical) *denotational semantics* for predicate logic, a variety of richer formal languages and (at least) fragments of natural language.

Consider, for example, the arithmetical sentences

$$\theta \equiv 1 + 0 = 1, \tag{1}$$

$$\eta \equiv \text{there exist infinitely many primes,} \tag{2}$$

$$\rho \equiv \text{there exist infinitely many twin primes.} \tag{3}$$

The first two are true, the third is a famous open problem. Surely we understand all three and we understand them differently, even as we recognize the first two to be true and concede that we do not know the truth value of the third. Frege would say that they mean different things because they express different senses, but he did not define sense: what exactly *is* the sense of θ and how does it

[1]During the preparation of this paper the author was partially supported by an NSF grant.
[2]I am grateful to Pantelis Nicolacopoulos for steering me to the van Heijenoort papers, to Alonzo Church for a copy of his still unpublished [1], to Tony Martin and the two referees for their useful remarks and especially to Kit Fine, whose extensive comments on a preliminary draft of the paper were very valuable and influenced substantially the final exposition.

differ from that of η? Our aim here is to propose a precise (mathematical, set theoretic) definition of Fregean sense and to begin the development of a rigorous *sense semantics* which explains differences in meaning like that between θ and η. The mathematical results of the paper are about formal languages, but they are meant to apply also to those fragments of natural language which can be formalized, much as the results of denotational semantics for formal languages are often applied to fragments of natural language. In addition to the language of predicate logic whose sense semantics are fairly simple, the theory also covers languages with description operators, arbitrary connectives and modal operators, generalized quantifiers, indirect reference and the ability to define their own truth predicate.

To explain the basic idea, consider a typical sentence of predicate logic like the arithmetical sentences above, but simpler:

$$\chi \equiv (\forall \mathrm{x})(\exists \mathrm{y})\mathrm{R}(\mathrm{x},\mathrm{y}) \vee (\exists \mathrm{z})\mathrm{Q}(\mathrm{z}). \tag{4}$$

Here R and Q are interpreted by relations R, Q on some domain A and we may suppose at first, for simplicity, that A is finite. If we know nothing about R and Q but their extensions, there is basically only one way to determine the truth value of χ, by the following procedure.

Step (1). Do steps (2) and (4). If one of them returns the value **t**, give the value **t**; if both of them return the value **f**, give the value **f**.

Step (2). For each $a \in A$, do step (3). If for every $a \in A$ the value **t** is returned, return the value **t**; if for some $a \in A$ the value **f** is returned, return the value **f**.

Step (3). Given $a \in A$, examine for each $b \in A$ the value $R(a, b)$. If one of these values is **t**, return the value **t**; if all these values are **f**, return the value **f**.

Step (4). For each $c \in A$, examine the value $Q(c)$. If one of these values is **t**, return the value **t**; if all these values are **f**, return the value **f**.

In contemporary computing terms, we have defined an *algorithm which computes the truth value of* χ. What if A is infinite? An attempt to "execute" (implement) these instructions will lead now to an infinitary computation, but still, the instructions make sense and we can view them as defining an ideal, infinitary algorithm which will compute ("determine" may be better now) the truth value of A. This algorithm is the *referential intension* or just **intension** of χ,

$$int(\chi) = \text{ the algorithm which computes the truth value of } \chi, \tag{5}$$

and we propose to model the sense of χ by its referential intension.

Plainly, we mean to reduce the notion of "sense" to that of "algorithm" and we must specify which precise notion of algorithm we have in mind. We will put this off until Section 3, where we will also review briefly the definitions and results

we need from the *theory of recursive algorithms*[3] developed in [22, 23, 24]. From this simple example, however, we can already see two basic properties algorithms must have if we are to use them in this way.

1.1. *Algorithms are semantic* (mathematical, set theoretic) *objects.* This is almost self evident, but we list it because there is some general tendency in the literature to confuse algorithms with *programs*, like the instructions above. A program is a piece of text, it means nothing uninterpreted; its interpretation (or one of them) is precisely the algorithm it defines, and that algorithm is no longer a syntactic object. On the account we are giving, the relation between a program and the algorithm it defines is exactly that between a sentence and its sense.

1.2. *Algorithms are relatively effective*, but they need not be effective in an absolute sense. Consider step (4) above which calls for the examination of the entire extension of the relation Q and cannot be "implemented" by a real machine if A is infinite. It is a concrete, precise instruction and *it is effective relative to the existential quantifier*, which we surely take as "given" when we seek the sense of sentences of predicate logic. Put another way, the claim that we understand χ entails that we understand the quantifiers which occur in it and the specific instructions of the intension simply "codify" this understanding in computational terms. Effective computability relative to assumed, given, infinitary objects has been studied extensively, especially in the work of Kleene in *higher type recursion.*

There is a third, fundamental property of algorithms we need which will be easier to explain after we discuss the first of three applications we intend to make of the proposed modeling of sense.

1.3. Self referential sentences. One of the main reasons for introducing sense is so that we may assign meaning to non-denoting names. Frege makes it clear that sentences may also fail to denote, but he only gives examples where the lack of a truth value is due to a non-denoting constituent proper name. There is, however, another stock of such examples in the self-referential sentences which are usually studied in connection with developing a *theory of truth.* Consider the following bare bones version[4] of the *liar* which avoids the use of a truth predicate,

$$\text{it is not the case that (6)}, \tag{6}$$

and the corresponding version of the *truthteller*,

$$\text{it is the case that (7)}. \tag{7}$$

Here we understand "it is the case" and "it is not the case" as direct affirmation and negation, i.e., with less regard for readability we could rewrite these definitions

[3]More accurately, we should call it the *theory of pure, single-valued, recursive algorithms*, since [22, 23, 24] do not deal with interaction and non-determinacy. It is possible to extend the results of this theory to algorithms whose implementations start engines, send and receive messages and "choose" in arbitrary ways which actions to execute, but for our present purpose pure algorithms suffice.

[4]These examples are alluded to in Kripke [19].

simply as

$$\neg(8), \tag{8}$$

$$(9). \tag{9}$$

These sentences have no problematic constituents and appear to be meaningful, we certainly understand at least the *liar* well enough to argue that it cannot have a truth value. In a Fregean approach we would expect both of them to have sense, and in fact different senses, since we understand them differently. A natural way to read this version of the *liar* (6) as an algorithm for computing its truth value leads to the single instruction

Step (1). Do step (1); if the value **t** is returned, give the value **f**; if the value **f** is returned, give the value **t**.

Similarly understood, the *truthteller* leads to the instruction

Step (1). Do step (1) and give the value returned.

The circular nature of these instructions corresponds to the self reference of the sentences, but there is nothing unusual about circular clauses like these in programs. The algorithms defined by them are *recursive algorithms* and (in this case, as one might expect), they do not compute any value at all, they *diverge*: on this account, neither the *liar* nor the *truthteller* have a truth value. On the other hand they have distinct referential intensions, because the instructions above define different algorithms.

To define the referential intension of self referential sentences we need recursive algorithms. We state this condition somewhat cryptically here; it will be explained in Section 3.

1.4. *Recursive definitions define algorithms directly.*

Kripke [19] has developed a rigorous semantics (with *truth gaps*) for a formal language which can define its own truth predicate, so that it can express indirect self reference and the more familiar versions of the *liar* and the *truthteller*. We will introduce in Section 2 the language LPCR of *Lower Predicate Calculus with Reflection* (or self reference), a richer and easier to use version of the language of Kripke which we will then adopt as the basic vehicle for explaining referential intensions. The denotational semantics of LPCR are Kripke's *grounded semantics*, by which all reasonable expressions of the *liar* and the *truthteller* fail to denote and cannot be distinguished. On the other hand, every sentence of LPCR (and a fortiori the less expressive language of Kripke) will be assigned a recursive algorithm for its referential intension, a Fregean sense which distinguishes these sentences from one another.[5] Part of the interest in this application is that it "explains"

[5]Calling this notion of sense for LPCR "Fregean" stretches the point a bit, since Frege counted non-denoting signs an imperfection of natural language and wished them banished from the language of mathematics. Nevertheless, we will see that its properties match quite closely the basic properties of sense assumed by Frege.

truth gaps in languages with self reference in terms of non-convergence of computations, a phenomenon which is quite common and well understood in the theory of algorithms.

1.5. Faithful translations. There is a special word in Greek, "μπατζανάκηδες", for the important relationship between two men whose wives are sisters. Most speakers of both Greek and English would agree that

"Niarchos and Onassis married sisters" (10)

is a faithful translation of

"Ο Νιάρχος και ο Ωνάσσης ήταν μπατζανάκηδες", (11)

even if they had never heard of Niarchos or Onassis and knew nothing about their wives; and no reasonable person would claim that

"Niarchos and Onassis were rivals" (12)

is a faithful translation of (11), although (as it happens) it is true, as is (11), and it is much closer in grammatical structure to (11) than (10)—"were" literally translates "ήταν". What makes (10) a faithful translation of (11) is that they both express the same algorithm for determining their truth: check to see if at any time Niarchos and Onassis were married to two sisters. On this account, we can define *a faithful translation* (at the sentence level) as *an arbitrary syntactic transformation of sentences from one language to another which preserves referential intension*, and that would apply even to languages with radically different syntax and distinct primitives, provided only that they both have a grammatical category of "assertoric sentence." The idea is direct from Frege who says that "the same sense has different expressions in different languages, or even in the same language" and that "the difference between a translation and the original should properly [preserve sense]."

Some claim that faithful translation is impossible, though I have never understood exactly what this means, perhaps because I was unable to translate it faithfully into Greek. In any case, the precise definition proposed here provides a technical tool for investigating the question rigorously. It may be that only fragments of Greek can be translated faithfully into English, or that we can only find translations which preserve *some* properties of the referential intension, and then we could study just what these fragments and these properties are. A typical application to formal languages is that Kripke's language can be translated faithfully into LPCR but not conversely, essentially because direct self reference is expressible in LPCR while Kripke's language allows only indirect self reference. The languages are denotationally equivalent (on most interesting structures), i.e., the same relations are definable by their formulas.

1.6. Primitives with complex sense. In defining the appropriate structures for sense semantics, we will allow the primitives of a language to express arbitrary senses which may be complex, as we assumed above to be the case with "μπατζανάκηδες" in Greek. This is technically convenient, both in developing a practical theory of translation and also for dealing with definitions: if we get tired of referring to Onassis and Niarchos as "married to sisters", we may want to add

the word "sisbands" to English with the stipulation that it expresses the same thing as "married to sisters", and then it will be a primitive of the new, extended English with a complex sense. We do this all the time both in natural and in formal languages. On the other hand, there are accounts of language which deal with this phenomenon in other ways and insist that the "ultimate" (atomic) primitives of a language cannot mean anything complex. It has also been argued that there is no coherent way to assign a non-trivial sense to some historical proper names like "Odysseus"or "Aristotle", cf. Kripke [18] and Donnellan [7]. If some of the primitives of the language stand for such atomic relations or directly denoting proper names, then in the modeling proposed here they will be assigned a trivial sense which is completely determined by their denotation. Our aim in this paper is to propose a logical mechanism which explains how complex senses can be computed from given (complex or simple) senses and there is nothing in the arguments and results of this paper which favors or contradicts the direct reference theory for names, or any theory of logical atomism for language.

1.7. Sense identity and intensional logic. Granting any precise, semantic definition of the sense of sentences of some language on some structure $\mathbf{A}$, put

$$\phi \sim_{\mathbf{A}} \psi \iff sense_{\mathbf{A}}(\phi) = sense_{\mathbf{A}}(\psi). \tag{13}$$

Do we always have

$$(\phi \ \& \ \psi) \sim_{\mathbf{A}} (\psi \ \& \ \phi) \quad \text{or} \quad \neg\neg\phi \sim_{\mathbf{A}} \phi?$$

Is the relation $\sim_{\mathbf{A}}$ of sense identity on a fixed structure decidable? Can we give a useful axiomatization for the class of sense identities valid in all structures and is this class decidable? Once we identify sense with referential intension, these questions become precise mathematical problems and they can be solved. The answers (even for a restricted language) will help us sharpen our intuition about sense, particularly as Frege is apparently quite obscure on the question of sense identity, cf. [29]. Incidentally, as one might expect, $(\phi \ \& \ \psi)$ and $(\psi \ \& \ \phi)$ are intensionally equivalent with the standard interpretation of & but $\neg\neg\phi$ has a more complex intension than ϕ.

Previous Work

Where sense is defined in the literature, it is often identified with some version of *proposition* in the sense of Carnap, i.e., the sense of ϕ is the function which assigns to each possible world W the truth value of ϕ in W. For example, Montague [20] adopts this definition. This does not explain the difference in meaning between arithmetic sentences like θ and η above which presumably have the same truth value in all possible worlds.

Church [2, 3, 4, 5] makes (essentially) two proposals for axiomatizing the relation of sense identity without giving a direct, semantic definition of sense. It is hard to compare such approaches with the present one, because they do not allow for specific, semantic import into the notion of sense, sense by them depends only on syntactic form. For example, according to Frege "John loves Mary" and "Mary is loved by John" should have the same sense. In a technical formulation,

this should mean that if two basic relation symbols R and Q are interpreted by (senseless) converse relations R and Q in a structure so that

$$R(x,y) \iff Q(y,x),$$

then for any two terms a, b, the sentences R(a,b) and Q(b,a) have the same sense, and they certainly have the same referential intension by our definition. This, however cannot be deduced from their syntactic form.

Although they work in different contexts and they have different aims, Church in the more recent [1], Cresswell [6] and van Heijenoort [30] use variants of the same idea for getting the sense of a sentence, by replacing in it all the primitive syntactic constructs by their denotations. For example, the Cresswell sense and the Church *propositional surrogate* of R(a, b) are (essentially) the triple $\langle R, a, b\rangle$. Both Church and Cresswell worry about introducing semantic import into meanings, but (if I understand them correctly), both would assign to R(a,b) and Q(b,a) the distinct triples $\langle R, a, b\rangle$ and $\langle Q, b, a\rangle$. I cannot find in such mechanisms a general device which can catch accidental, complex semantic relationships like that between a relation and its converse.

On the other hand, the idea that the sense of a sentence determines its denotation is at the heart of the Frege doctrine and practically everyone who has written on the matter uses some form of computational analogy to describe it. Perhaps closest to the present approach is Dummett's interpretation of Frege, most clearly quoted by Evans in a footnote of [9]:

> "This leads [Dummett] to think generally that the sense of an expression is (not a way of thinking about its [denotation], but) a method or procedure for determining its denotation. So someone who grasps the sense of a sentence will be possessed of some method for determining the sentence's truth value.... The procedures in question cannot necessarily be effective procedures, or procedures we ourselves are capable of following."

Evans goes on to call this view "ideal verificationism" and says that "there is scant evidence for attributing it to Frege." Dummett himself says in [8] that

> "[Frege's basic argument about identity statements] would be met by supposing the sense of a singular term to be related to its reference as a programme to its execution, that is, if the sense provided an effective procedure of physical and mental operations whereby the reference would be determined."

If we replace "programme" and "execution" by "algorithm" and "value", then this comes very close to identifying sense with referential intension. What we add here, of course is a specific, mathematical notion of recursive algorithm which makes it possible to develop a rigorous theory of sense.

Outline of what follows

We will state the basic definitions in the body of the paper for the simple to describe language LPCR and we will include enough expository material about

the theory developed in [22, 23, 24] so that the gist of what we are saying can be understood without knowledge of these papers. On the other hand, the natural domain for this theory is the *Formal Language of Recursion* FLR defined in [23] and in Section 4 we will prove the main result of the paper for this general case. This is *the decidability of intensional identity for terms of* FLR*, on any fixed structure*, provided only that the signature (the number of primitives) is finite. Intensional identity is relatively trivial on sentences of predicate logic, but we will establish a non-trivial lower bound for its (decidable) degree of complexity for sentences of LPCR: as one might expect, once you are allowed self reference you can say the same thing in so many different ways, that it is not always obvious that it is still the same thing. Finally, the proof of decidability for intensional identity has independent interest and it gives some insight into the concept of a faithful translation. It also suggests an axiomatization for the *logic of intensions*, but we will not consider this here.

§2. Languages with direct self reference.

A *relational* (first order) *signature* or *similarity type* τ is any set of formal *relation* (or predicate) *symbols*, each equipped with a fixed, integer (≥ 0) *arity*. We assume the usual inductive definition of *formulas* for the language $\mathrm{LPC} = \mathrm{LPC}(\tau)$ of *Lower Predicate Calculus* (with identity) on the signature τ, with formal individual variables v_1, v_2, ... and (for definiteness) logical symbols $=$, $\neg$, $\&$, $\vee$, $\supset$, $\exists$, $\forall$. To obtain the language $\mathrm{LPCR} = \mathrm{LPCR}(\tau)$ of *Lower Predicate Calculus with Reflection* (or self reference) on the same signature, we first add to the basic vocabulary an infinite sequence P_1^k, P_2^k, ... of formal k-ary *partial relation variables*, for each $k \geq 0$. The number k is the *arity* of P_i^k. We call these "partial" relation variables because we will interpret them by partially defined relations in the intended semantics, but syntactically they are treated like ordinary relation variables, so we add to the formation rules for formulas in LPC the following:

2.1. If P is a partial relation variable of arity k and x_1, ..., x_k are variables, then the string $P(x_1, \ldots, x_k)$ is a formula with free variables x_1, ..., x_k and P. In particular, for every *partial, propositional variable* P with arity 0, $P(\,)$ is a formula.

Self reference is introduced in LPCR by the following key, last rule of formula formation.

2.2. If ϕ_0, ..., ϕ_n are formulas, P_1, ..., P_n are partial relation variables and for each i, $\vec{u}^i$ is a sequence of individual variables of length the arity of P_i, then the string

$$\phi \equiv \phi_0 \text{ where } \{P_1(\vec{u}^1) \simeq \phi_1,\ \ldots,\ P_n(\vec{u}^n) \simeq \phi_n\} \tag{14}$$

is a formula. The bound occurrences of variables in ϕ are the bound occurrences in *the head* ϕ_0 and *the parts* ϕ_1, ..., ϕ_n of ϕ, all the occurrences of P_1, ..., P_n and the occurences of the individual variables of each $\vec{u}^i$ in $P_i(\vec{u}^i) \simeq \phi_i$.

Notice that the sequence $\vec{u}^i$ must be empty here when P_i is a partial, propositional variable. The simplest examples of self referential sentences involve such "propositional reflection," e.g., the formal LPCR versions of the direct *liar* and *truthteller* intended by (6) and (7) are:

$$\mathit{liar} \equiv P(\,) \text{ where } \{P(\,) \simeq \neg P(\,)\}, \qquad \mathit{truthteller} \equiv P(\,) \text{ where } \{P(\,) \simeq P(\,)\}. \tag{15}$$

The more complex

$$\mathit{liar}' \equiv (\exists x)P(x) \text{ where } \{P(x) \simeq \neg P(x)\} \tag{16}$$

asserts that "the relation which is always equivalent to its own negation holds of some argument," and it too will have no truth value.

In the definition of (denotational) semantics for LPCR which follows, we will adopt a general, somewhat abstract view of formal language semantics which is unnecessarily complex for predicate logic and just barely useful for LPCR. The extra effort is worth it though, because this formulation of semantics generalizes immediately to a large class of languages (including rich fragments of natural language) and is indispensable for the description of referential intensions in the next section, even for the case of LPC.

2.3. Partial functions. Let us first agree to view an m-ary relation on a set X as a function

$$R : X^m \to TV, \qquad (TV = \{\mathbf{t}, \mathbf{f}\}) \tag{17}$$

which assigns a truth value to each m-tuple in X. A partial m-ary relation is then a *partial function*

$$P : X^m \rightharpoonup TV, \tag{18}$$

i.e., a function defined on some of the m-tuples in X with range TV—notice the symbol "$\rightharpoonup$" which suggests that for some $\vec{x}$, $P(\vec{x})$ may be undefined. In dealing with partial functions, we will use the standard notations

$$\begin{aligned} f(\vec{x})\downarrow \quad &\Longleftrightarrow \quad \vec{x} \text{ is in the domain of } f, \\ f(\vec{x}) \simeq w \quad &\Longleftrightarrow \quad f(\vec{x})\downarrow \text{ and } f(\vec{x}) = w, \\ f \subseteq g \quad &\Longleftrightarrow \quad (\forall \vec{x}, w)[f(\vec{x}) \simeq w \implies g(\vec{x}) \simeq w], \end{aligned}$$

and we assume the usual (strict) composition rule so that (for example)

$$f(g(\vec{x}), h(\vec{x}))\downarrow \Longrightarrow g(\vec{x})\downarrow \text{ and } h(\vec{x})\downarrow . \tag{19}$$

We let

$$\emptyset = \text{the totally undefined partial function.} \tag{20}$$

In set theoretic terms this is the empty set of ordered pairs and it is the least partial function in the partial ordering $\subseteq$.

2.4. Structures for LPCR. We interpret LPCR in the usual structures of predicate logic, i.e., tuples of the form $\mathbf{A} = (A, R_1, \ldots, R_l)$, where A is some non-empty set and each relation R_j interprets the corresponding formal relation symbol R_j of the signature. With the convention just described, $\mathbf{A}$ is simply a sequence of functions on its universe A, with values in TV. Of course we need more than these

functions to understand the meaning of formulas in $\mathbf{A}$, we also need the meaning of $=$, $\neg$, $\forall$, etc. These are taken for granted, but if we include them and the set of truth values TV for completeness, we get the tuple

$$\mathbf{A} = (TV, A, R_1, \ldots, R_l, eq_A, \neg, \&, \vee, \supset, \exists_A, \forall_A). \tag{21}$$

Now

$$eq_A(x,y) = \begin{cases} \mathbf{t}, \text{ if } x = y, \\ \mathbf{f}, \text{ if } x \neq y, \end{cases} \qquad \neg(a) = \begin{cases} \mathbf{f}, \text{ if } a = \mathbf{t}, \\ \mathbf{t}, \text{ if } a = \mathbf{f} \end{cases} \tag{22}$$

are clearly (total) functions on A and the set of truth values TV. The objects $\exists_A$, $\forall_A$ are *functionals*, partial functions which take unary, partial relations as arguments:

$$\exists_A(P) \simeq \begin{cases} \mathbf{t}, \text{ if for some } x \in A, \ P(x) \simeq \mathbf{t}, \\ \mathbf{f}, \text{ if for all } x \in A, \ P(x) \simeq \mathbf{f}, \end{cases} \tag{23}$$

$$\forall_A(P) \simeq \begin{cases} \mathbf{t}, \text{ if for all } x \in A, \ P(x) \simeq \mathbf{t}, \\ \mathbf{f}, \text{ if for some } x \in A, \ P(x) \simeq \mathbf{f}. \end{cases} \tag{24}$$

These functionals represent the quantifiers because for every unary partial relation P,

$$(\exists x \in A)[P(x) \simeq w] \iff \exists_A(P) \simeq w,$$

and more generally, on partial relations of any arity,

$$(\exists x \in A)[P(x, \vec{y}) \simeq w] \iff \exists_A(\lambda(x)P(x, \vec{y})) \simeq w,$$

with the obvious meaning for the λ-operator on partial functions: the term $\exists_A(\lambda(x)P(x,\vec{y}))$ is equivalent to (takes exactly the same values as) the formula $(\exists x)P(x,\vec{y})$ on the set A.

Notice that $\forall_A(P)$ is not defined if P is a partial relation which never takes the value $\mathbf{f}$. However—and this is a basic property of these objects—both $\exists_A$ and $\forall_A$ are *monotone*, i.e.,

$$[\exists_A(P) \simeq w \ \& \ P \subseteq P'] \implies \exists_A(P') \simeq w,$$
$$[\forall_A(P) \simeq w \ \& \ P \subseteq P'] \implies \forall_A(P') \simeq w.$$

This is obvious from their definition.

One is tempted to interpret $\&$, $\vee$ and $\supset$ by the usual, total functions on TV, but this will not work if we want to apply them to partial relations; because if $\vee$ were a binary function, then by the rules of strict composition we would have $(P \vee Q)\downarrow \implies [P\downarrow \ \& \ Q\downarrow]$, while we certainly want $P \simeq \mathbf{t} \implies (P \vee Q) \simeq \mathbf{t}$, even when Q is undefined. We set instead:

$$P \vee Q \simeq \begin{cases} \mathbf{t}, \text{ if } P(\,) \simeq \mathbf{t} \text{ or } Q(\,) \simeq \mathbf{t}, \\ \mathbf{f}, \text{ if } P(\,) \simeq Q(\,) \simeq \mathbf{f}, \end{cases} \quad P \ \& \ Q \simeq \begin{cases} \mathbf{t}, \text{ if } P(\,) \simeq Q(\,) \simeq \mathbf{t}, \\ \mathbf{f}, \text{ if } P(\,) \simeq \mathbf{f} \text{ or } Q(\,) \simeq \mathbf{f}, \end{cases} \tag{25}$$

and similarly for $\supset$. Here P and Q are partial, propositional variables and clearly

$$[P \subseteq P', \ Q \subseteq Q', \ P \ \& \ Q \simeq w] \implies P' \ \& \ Q' \simeq w$$

and similarly with $\vee$, i.e., the propositional connectives are also monotone.

We now define precisely (partial, monotone) *functionals* and *functional structures*. The complication is due to the unavoidable fact that these are *typed objects*, never too simple to deal with.

2.5. Partial, monotone functionals. A (many sorted) *universe* is any collection of sets

$$\mathcal{U} = \{\mathcal{U}(\bar{u}) \mid \bar{u} \in B\}$$

indexed by some (non-empty) set B of *basic types*. Members of $\mathcal{U}(\bar{u})$ are of (basic) *type* $\bar{u}$ in $\mathcal{U}$. For each Cartesian product

$$U = \mathcal{U}(\bar{u}_1) \times \cdots \times \mathcal{U}(\bar{u}_n), \quad (\bar{u}_1, \ldots, \bar{u}_n \in B)$$

and each $W = \mathcal{U}(\bar{w})$, $\bar{w} \in B$, we let

$$P(U, W) = \{p \mid p : U \rightharpoonup W\}$$

be the set of all partial functions on U to W; the objects in $P(U, W)$ are of *partial function* or *pf type* $((\bar{u}_1, \ldots, \bar{u}_n) \rightharpoonup \bar{w})$ in $\mathcal{U}$. We allow U to be *the Cartesian product of no factors* I, in which case the objects of pf type $(() \rightharpoonup \bar{w})$ are the *partial constants* of type $\bar{w}$. A *point type* in B is any tuple $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_n)$, where each $\bar{x}_i$ is either a basic or a pf type in B and a *point of type* $\bar{x}$ is any tuple $x = (x_1, \ldots, x_n)$ where each object x_i has type $\bar{x}_i$ in $\mathcal{U}$. The (product) space of all points of type $\bar{x}$ is naturally partially ordered by

$$(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n) \iff x_1 \leq_1 y_1 \ \& \ \ldots \ \& \ x_n \leq_n y_n,$$

where $\leq_i$ is just the identity $=$ when $\bar{x}_i$ is a basic type and for a pf type $\bar{x}_i$, $\leq_i$ is the standard partial ordering on partial functions. Finally, a (partial, monotone) *functional* of type $(\bar{x} \rightharpoonup \bar{w})$ is any partial function $f : X \rightharpoonup W$ which is monotone in this partial ordering, i.e.,

$$[f(x) \simeq w \ \& \ x \leq x'] \Longrightarrow f(x') \simeq w.$$

Notice that pf types are also functional types and every partial function in $\mathcal{U}$ is a (degenerate) functional.

2.6. Functional structures. A *finite* (functional) *signature* is a triple $\tau = (B, \{\mathrm{f}_1, \ldots, \mathrm{f}_n\}, d)$, where $B = \{\mathsf{b}_1, \ldots, \mathsf{b}_k\}$ is a finite set of basic types, $\mathrm{f}_1, \ldots, \mathrm{f}_n$ are arbitrary function symbols and d is a mapping which assigns a functional type to each f_i; a *functional structure* of signature τ is a tuple

$$\mathbf{A} = (\mathcal{U}, f_1, \ldots, f_n) = (U_1, \ldots, U_k, f_1, \ldots, f_n) \tag{26}$$

where $\mathcal{U}$ is a universe over B with the basic sets $U_i = \mathcal{U}(\mathsf{b}_i)$, $i = 1, \ldots, k$, and each f_j is a functional on $\mathcal{U}$ of type $d(\mathrm{f}_j)$.

For example, this representation of a standard first-order structure $\mathbf{A}$ in (21) above has just two basic sets TV and A of respective basic types (say) bool and a. An n-ary partial relation P on A is a functional of type $((\mathsf{a}^n) \rightharpoonup \mathsf{bool})$ and similarly, an n-ary partial function $p : A^n \rightharpoonup A$ is of pf and functional type $((\mathsf{a}^n) \rightharpoonup \mathsf{a})$. The functional $\&$ if of type $((((\,) \rightharpoonup \mathsf{bool}), (() \rightharpoonup \mathsf{bool})) \rightharpoonup \mathsf{bool})$ and $\exists_A$ is of type $(((\mathsf{a}) \rightharpoonup \mathsf{bool}) \rightharpoonup \mathsf{bool})$, as is $\forall_A$. Not much is gained by computing these complex

types, but it is important to notice that we are dealing with typed objects.[6] We will concentrate on this concrete example for the rest of this section, but everything we say generalizes directly to arbitrary functional structures which can be used to model languages with additional propositional connectives, generalized quantifiers, modal operators and the like.

2.7. The denotational semantics of LPCR. Fix a structure $\mathbf{A}$ as in (21). With each formula ϕ and each sequence of (individual and partial relation) variables $\vec{\mathrm{x}} = \mathrm{x}_1, \ldots, \mathrm{x}_n$ which includes all the free variables of ϕ, we associate a (partial, monotone) functional

$$f_{\vec{\mathrm{x}},\phi} = den_{\mathbf{A}}(\vec{\mathrm{x}}, \phi) : X \rightharpoonup TV,$$

where X is the space of all n-tuples $(x_1, \ldots, x_n)$ such that the type of x_i matches that of the formal variable x_i. The definition is by induction on the length of ϕ and it is quite trivial, except for the case of the new primitive `where` of LPCR. For example, skipping the subscripts,

$$\left.\begin{array}{rcl} den(\vec{\mathrm{x}}, \mathrm{x}_i = \mathrm{x}_j)(\vec{x}) & = & eq_A(x_i, x_j), \\ den(\vec{\mathrm{x}}, \mathrm{R}_m(\mathrm{x}_i, \mathrm{x}_j, \mathrm{x}_k))(\vec{x}) & = & R_m(x_i, x_j, x_k), \\ den(\vec{\mathrm{x}}, \phi \vee \psi)(\vec{x}) & \simeq & \lambda(\,)den(\vec{\mathrm{x}}, \phi)(\vec{x}) \vee \lambda(\,)den(\vec{\mathrm{x}}, \psi)(\vec{x}), \\ den(\vec{\mathrm{x}}, (\exists \mathrm{y})\phi(\vec{\mathrm{x}}, \mathrm{y}))(\vec{x}) & \simeq & \exists_A(\lambda(y)den(\vec{\mathrm{x}}, \mathrm{y}, \phi(\vec{\mathrm{x}}, \mathrm{y}))(y, \vec{x})). \end{array}\right\} \quad (27)$$

Notice the use of the dummy λ operation in the case of $\vee$, which is needed to turn an element in TV (or "the undefined") into a partial, propositional constant. Proof that the operation defined in each of these cases is a (monotone) functional is routine, depending only on the trivial fact that the composition of monotone, partial functions is also monotone.

For the less trivial case of the `where` construct, suppose

$$\phi \equiv \phi_0 \text{ where } \{P_1(\vec{u}^1) \simeq \phi_1, \ \ldots, \ P_n(\vec{u}^n) \simeq \phi_n\}$$

and for each $\vec{x}$, use the induction hypothesis to define

$$f_{\vec{x},i}(\vec{u}^i, P_1, \ldots, P_n) \simeq den(\vec{\mathrm{u}}^i, \mathrm{P}_1, \ldots, \mathrm{P}_n, \phi_i)(\vec{u}^i, P_1, \ldots, P_n), \quad (i = 0, \ldots, n).$$

Now the functionals $f_{\vec{x},i}$ are all monotone and it follows from basic results in the theory of least-fixed-point recursion that the system of equations

$$P_i(\vec{u}^i) \simeq f_{\vec{x},i}(\vec{u}^i, P_1, \ldots, P_n), \quad (i = 1, \ldots, n) \quad (28)$$

has a (unique) sequence of simultaneous *least* (as partial functions) solutions $\overline{P}_{\vec{x},i}$, $i = 1, \ldots, n$. We set

$$den(\vec{\mathrm{x}}, \phi)(\vec{x}) \simeq f_{\vec{x},0}(\overline{P}_{\vec{x},1}, \ldots, \overline{P}_{\vec{x},n}). \quad (29)$$

We need to verify that this is a functional again, i.e., monotone, but the proof is quite direct.

The denotation of a sentence (with no free variables) is independent of its arguments and we set

$$value(\phi) \simeq den(\phi)(\,). \quad (30)$$

[6] If both A and $P(A, A)$ are among the basic sets in a universe with respective basic types a and paa, then a partial function $f : P(A, A) \rightharpoonup A$ may be viewed as either of pf type $(\mathsf{paa} \rightharpoonup \mathsf{a})$ or (if it is monotone), as a functional, of type $((\mathsf{a} \rightharpoonup \mathsf{a}) \rightharpoonup \mathsf{a})$,

There is a lot of notation here and an appeal to results which are not as widely known as they might be, but the underlying idea is very simple. Consider the *liar* defined by (6). The definition gives

$$value(liar) \simeq \overline{P}(\,),$$

where $\overline{P}$ is the least solution of the equation

$$P(\,) \simeq \neg(P(\,));$$

since the totally undefined $P = \emptyset$ satisfies this equation, the *liar* has no truth value and neither does the *truthteller*. For *liar*$'$ we need the least solution of the equation

$$P(x) \simeq \neg P(x)$$

which is again $\emptyset$, so that it too receives no truth value.

Consider also the following two versions of the *ancestral*, in which we will use the notation

$$x \prec y \iff x \text{ is a child of } y$$

and we take I and *Euclid* to be individual constants with fixed references:

$$\phi \equiv P(I) \text{ where } \{P(x) \simeq x \prec \mathit{Euclid} \vee (\exists y)[x \prec y \;\&\; P(y)]\}, \tag{31}$$

$$\psi \equiv Q(\mathit{Euclid}) \text{ where } \{Q(z) \simeq I \prec z \vee (\exists y)[y \prec z \;\&\; P(y)]\}. \tag{32}$$

Both ϕ and ψ express the intended assertion

I am a descendant of Euclid,

but in different ways, intuitively

ϕ : I am a child of Euclid or the child of a descendant of Euclid,

ψ : Euclid is my father or the father of an ancestor of mine.

It is easy to verify that both ϕ and ψ take the value **t** if, indeed, I am a descendant of Euclid. In the more likely, opposite case, it can be verified that ϕ will take the value **f** only if there is an upper bound to the length of ascending ancestral chains of the form

$$I \prec y_1 \prec y_2 \prec \cdots \prec y_n,$$

while ψ will be defined and false only when there is an upper bound to the length of descending chains of parenthood of the form

$$\mathit{Euclid} \succ z_1 \succ z_2 \succ \cdots \succ z_n.$$

Both of these conditions are evidently true, but the first one undoubtedly claims a bit more about our world (and the meaning of the relationship "child") than the relatively innocuous second one.

2.8. Definability in LPCR. A (possibly partial) relation $R \subseteq A^n$ is *weakly representable* in LPCR if there is a formula ϕ with free variables $\mathrm{x}_1, \ldots, \mathrm{x}_n$ such that

$$R(\vec{x}\,) \iff den(\vec{\mathrm{x}}, \phi)(\vec{x}\,) \simeq \mathbf{t},$$

and *strongly representable* or just *definable* in LPCR if we have

$$R(\vec{x}\,) \simeq den(\vec{\mathrm{x}}, \phi)(\vec{x}\,).$$

Notice that strongly representable total relations are defined by *total formulas*, with no truth gaps. Those familiar with the theory of *inductive definability* [22] can verify without much difficulty that *a total relation on A is* LPCR-*weakly representable just in case it is absolutely inductive in* $\mathbf{A}$ and it is LPCR-*strongly representable just in case it is absolutely hyperelementary in* $\mathbf{A}$. We will not pursue here this connection or the relation of LPCR with the several languages with fixed-point operators studied in logic and theoretical computer science.

Suppose now that the structure $\mathbf{A}$ is infinite and *acceptable* in the sense of [21], so that we can code tuples from A by single elements of A and we can manipulate these tuples in LPCR.[7] We can then define a copy of the integers in A and we can assign *Gödel numbers* to formulas of LPCR and *Gödel codes* to sentences in the expanded language, where we add a formal name for each element of A. Finally, we can define the *partial truth predicate* of the structure,

$$T_{\mathbf{A}}(s) \simeq \begin{cases} \mathit{value}(\theta_s), \text{ if } s \text{ is the Gödel code of a sentence } \theta_s, \\ \mathbf{f}, \text{ otherwise.} \end{cases} \tag{33}$$

2.9. THEOREM. *If* $\mathbf{A}$ *is acceptable, then its partial truth predicate* $T_{\mathbf{A}}$ *for* LPCR *is definable in* LPCR.

The proof of this is neither difficult nor surprising: we simply express the inductive definition of $T_{\mathbf{A}}(s)$ directly by a `where` formula of LPCR.

2.10. Kripke's language. Kripke [19] considers the formal extension of the language LPC of predicate logic by a single unary relation symbol T which is meant to denote (on acceptable structures) the truth predicate of the extended language containing it. To define the interpretation T of T (on Gödel codes of sentences), he considers the natural inductive conditions that T must satisfy and then takes the least[8] partial relation which satisfies these conditions; the truth value of a sentence θ then (when it has one) is the value $T(s)$ of T on the Gödel code s of θ. It is quite routine to define a recursive mapping

$$\phi \mapsto \tau\rho(\phi) \tag{34}$$

which assigns to each formula ϕ of the language of Kripke a formula $\tau\rho(\phi)$ of LPCR which "means the same thing," and in particular has the same denotation: basically, we replace each occurrence of T by the LPCR formula which defines $T_{\mathbf{A}}$ on the appropriate argument. In particular, the usual versions of the *liar* and other "paradoxical" sentences which employ indirect self reference are expressible in LPCR and are assigned denotations in accordance with the grounded semantics of Kripke.

[7]The precise definition of *acceptability* is not important. Kripke [19] points out that much weaker hypotheses will do, actually all we need is the existence of a first-order-definable, one-to-one function $\pi : A \times A \to A$.

[8]These are Kripke's *grounded semantics*, which are most relevant to what we are doing here. We will discuss briefly alternative fixed-point semantics in the next section.

2.11. Descriptions. Suppose we now expand LPCR by a construct of the form

$$\phi \mapsto (\mathsf{the}\ \mathrm{x})\phi(\mathrm{x}) \tag{35}$$

which assigns a term to every formula $\phi(\mathrm{x})$, where the variable x may or may not occur free in $\phi(\mathrm{x})$. We understand (35) as a new clause in the recursive definition of terms and formulas, by which terms can occur wherever free variables can occur.

Notice first that the where construct gives an easy mechanism to control the *scoping* of descriptions. For example, Russell's "bald King of France" example in [26] is best formalized by

$$BKF \equiv \text{The King of France is bald} \equiv P((\mathsf{the}\ \mathrm{x})KF(\mathrm{x}))\ \mathsf{where}\ \{P(\mathrm{x}) \simeq B(\mathrm{x})\}, \tag{36}$$

whose reformulation according to the theory of descriptions

$$BKF \iff (\exists!\mathrm{x})KF(x)\ \&\ (\exists\mathrm{x})[KF(\mathrm{x})\ \&\ B(\mathrm{x})] \tag{37}$$

is false, as Russell would wish it. If $B(\mathrm{x})$ is a complex expression, as it most likely is, Russell's analysis of the simpler $B((\mathsf{the}\ \mathrm{x})KF(\mathrm{x}))$ might turn out to be true or false depending on the particular form of the formula $B(\mathrm{x})$.

We naturally wish to set

$$den(\vec{\mathrm{y}}, (\mathsf{the}\ \mathrm{x})\phi(\mathrm{x}))(\vec{y}) \simeq the_A(\lambda(x)den(\vec{\mathrm{y}}, \mathrm{x}, \phi(\mathrm{x}))(\vec{y}, x) \tag{38}$$

with a suitable functional

$$the_A : P(A, TV) \rightharpoonup A, \tag{39}$$

and it is clear that the_A should satisfy

$$[(\forall x)P(x)\!\downarrow\ \&\ (\exists!x)P(x) = \mathbf{t}] \Longrightarrow the_A(P) \simeq \text{ the unique } x \text{ such that } P(x) = \mathbf{t}. \tag{40}$$

It is also clear that (to insure monotonicity) $the_A(P)$ cannot be always defined, e.g., $the_A(\emptyset)$ will take no value. The question is how to set $the_A(P)$ when we know from the values of P that no $P' \supseteq P$ defines a single object, i.e., when the following holds:

$$Error(P) \iff (\forall x)[P(x) \simeq \mathbf{f}] \vee (\exists x)(\exists y)[x \neq y\ \&\ P(x) \simeq P(y) \simeq \mathbf{t}]. \tag{41}$$

There are three plausible ways to proceed and we can label them by the denotational semantics for descriptive phrases which they generate.

2.12. Frege's solution. If $Error(P)$, we let $the_A(P)$ stay undefined. Russell argues eloquently against this solution which would also leave BKF without a truth value. Now Russell's—and everybody else's theory—would certainly leave undefined the term

$$(\mathsf{the}\ \mathrm{x})[[\mathrm{x} = 1\ \&\ liar] \vee [\mathrm{x} = 0\ \&\ \neg liar]], \tag{42}$$

which we do not know how to handle, but it does not seem right to have $the_A(P)$ undefined when $Error(P)$ holds and we know exactly what is wrong.

2.13. Russell's solution. We add to the basic set A a new element e_A, we decree that

$$Error(P) \Longrightarrow the_A(P) \simeq e_A, \tag{43}$$

and we extend all partial relations so that

$$x_i = e_A \Longrightarrow P(x_1, \ldots, x_n) \simeq \mathbf{f}, \quad (i = 1, \ldots, n). \tag{44}$$

Notice that this makes $the_A(P)$ always defined on total functions P, and (easily) it assigns the same truth values to sentences as Russell's theory of descriptions.

2.14. The programmer's solution. In an effort to test Russell's solution by the experimental method, I posed directly to four friends the hot question: *do you think that the King of France is bald?* The one who knew too much responded "*Russell thought not*," but the other three gave me exactly the same, predictable answer: "*there is no King of France.*" To make sense out of this thinking, we add a new element $\mathbf{e}$ (for "error") to the set of truth values TV, we decree (43) as above and we extend all partial relations so that

$$x_i = e_A \Longrightarrow P(x_1, \ldots, x_n) \simeq \mathbf{e}. \quad (i = 1, \ldots, n). \tag{45}$$

We must also extend the connectives and quantifiers in the obvious way, to take account of the new truth value, e.g.,

$$\mathbf{t} \;\&\; \mathbf{e} = \mathbf{e}, \quad \mathbf{t} \vee \mathbf{e} = \mathbf{t}, \quad \neg\mathbf{e} = \mathbf{e},$$

$$\exists_A(P) \simeq \begin{cases} \mathbf{t}, \text{ if for some } x \in A,\ P(x) \simeq \mathbf{t}, \\ \mathbf{f}, \text{ if for all } x \in A,\ P(x) \simeq \mathbf{f}, \\ \mathbf{e}, \text{ if } P \text{ is total },\ (\forall x \in A)[P(x) \neq \mathbf{t}],\ (\exists x \in A)[P(x) = \mathbf{e}]. \end{cases}$$

This is a refinement of the Russell solution: when we know that there is no good way to interpret a descriptive phrase, we record an "error" and we let the Boolean operations propagate the error when they receive it from an argument they need. We might in fact enrich the set TV with a whole lot of distinct "error messages" which explain just what went wrong with the descriptions we attempted to compute, one of these error messages presumably being "there is no King of France".

These functional interpretations of the yield denotational semantics for descriptive phrases which make it possible to view (the x)ϕ(x) as a constituent of every sentence θ in which it occurs while assigning to θ the correct (expected) denotation. Of course, denotations alone cannot account for the difference in meaning between "*Scott is the author of Waverly*" and "*Scott is Scott*", both of them being true by any of the proposed solutions. These sentences mean different things because they have distinct referential intensions, the first calling for us to check whether Scott indeed was Scott (a trivial algorithm) while the second requires that we verify whether Scott wrote the Waverly novels. We will define the referential intensions of descriptive phrases in the next section.

2.15. The Formal Language of Recursion FLR. In [23] we associated with each functional signature τ the *Formal Language of Recursion* FLR = FLR(τ), a language of terms which in addition to the function symbols $\mathrm{f}_1, \ldots, \mathrm{f}_n$ of τ has basic and partial function variables over the types of τ and the additional type bool (standardly interpreted), constants $\mathbf{t}$ and $\mathbf{f}$ naming themselves, a *conditional construct* and a general where construct which allows recursive definitions in all basic types. Quite obviously, we can view LPC, LPCR and its natural extensions

as fragments of FLR, with special signatures and some "sugaring" of the notation; and the denotational semantics we defined are special cases of the denotational semantics for FLR, which are treated in [23]. The mathematical theory of self-referential semantics, referential intension and sense is best developed for FLR and then applied to the specific cases of interest, but its usefulness and justification rest on these applications.

§3. Recursive algorithms and referential intensions.

The distinction between an *algorithm* f and *the object* $\overline{f}$ *computed by that algorithm*—typically a *function*—is well understood. Consider, for example the need to alphabetize (sort) a list of words in some alphabet which comes up in a myriad of computing applications, from compilers of programming languages to sophisticated word processing programs. There are hundreds of known *sorting algorithms* and the study of their properties is a sizable cottage industry straddling theoretical and applied computer science: they all compute the same *sorting function* which assigns to every list of words its ordered version.

Equally clear is the difference between an algorithm and its various *implementations* in specific machines or the *programs* which express it in specific programming languages. It is true that this distinction is sometimes denied by computer scientists who take a formalist or nominalist position towards the foundations of computing and recognize nothing but programs, their syntactic properties and the "symbolic computations" they define. Yet I have never known a programmer who did not know the difference between "programming the mergesort algorithm in LISP" (a relatively trivial task) and programming the mergesort in some assembly language, a notoriously difficult exercise in introductory programming courses. What can the last quoted sentence possibly mean if there is no such thing as "the mergesort algorithm"? The question is surely more serious than this flippant remark would suggest, as serious as the general question of the formalist and nominalist approaches to mathematics, but this is not the place to address it seriously. Suffice it to say that I am adopting a classical, realist view which takes it for granted that algorithms "exist" independently of the programs which express them, much as real numbers "exist" independently of the definitions which can be given for some of them in specific formal languages.[9]

Although the concept of algorithm is universally recognized as fundamental for computer science, it is traditional to treat it as a "premathematical" notion and avoid defining it rigorously. Where there is need for a precise definition, for example in complexity theory, it is generally assumed that algorithms are faithfully represented by *models of computation*, automata, Turing machines, random access machines and the like. We record here the most general definition of this type.

[9] A sequence of recent court decisions in patent law may possibly lead to a legal distinction between a program and the algorithm it expresses! Euclid could get a copyright on the *Elements*, which includes a definition of his famous algorithm for finding the greatest common divisor of two integers, but could he also patent the algorithm itself so that no one could use it without paying him a fee? Amazingly, the question came up in the courts in connection with the *fast multiplication algorithm*, which solves in a novel way a problem even more elementary than Euclid's.

3.1. An **iterator** (or *abstract sequential machine*) on the set X to W is a tuple

$$f = (S, T, \pi, in, out), \tag{46}$$

where the following conditions hold.

1. S is a non-empty set, the *states* of f, and $T \subseteq S$ is a designated subset of *terminal states*.

2. $\pi : S \to S$ is a total function, the *transition function* of f.

3. $in : X \to S$ and $out : T \to W$ are the *input* and *output* functions of f.

To compute with f on a given $x \in X$, we first find the initial state $in(x)$ associated with x and then we iterate the transition function beginning with this,

$$\pi^0(x) = in(x),\ \pi^1(x) = \pi(\pi^0(x)),\ \ldots, \pi^{n+1}(x) = \pi(\pi^n(x)),\ \ldots \tag{47}$$

until (and if) we find some n such that $\pi^n(x)$ is a terminal state; if such an integer exists, we set

$$N(x) \simeq (\text{the least } n)[\pi^n(x) \in T] \tag{48}$$

and we give as output the value of *out* on the terminal state $N(x)$,

$$\overline{f}(x) \simeq out(\pi^{N(x)}(x)). \tag{49}$$

Equations (47), (48), (49) taken together define the partial function $\overline{f} : X \rightharpoonup W$ *computed by the iterator* f.

In the most familiar example of a Turing machine with a one-way tape set up to compute a unary, partial function on the integers, states are triples (s, τ, i) where s is an internal state of the machine, τ is a description of the tape with finitely many (non-blank) symbols on it and i is the scanned square; the transition function π is determined by the finite table of the machine; (s, τ, i) is terminal if the machine halts in the situation described by it; for each integer n, $in(n) = (s_0, \tau_n, 0)$, where s_0 is the initial, internal state and τ_n is the tape with a string of $n+1$ consecutive 1's on it; and *out* counts the consecutive 1's on the tape and subtracts 1 from them.

Turing's compelling analysis of mechanical computation in [28] establishes beyond doubt that *every partial function on the integers which can be computed by a finitary, mechanical (deterministic) device can also be computed by a Turing machine*, what we now call *the Church-Turing Thesis*. His arguments also make it quite obvious that the more general iterators can represent faithfully all the intensional aspects of mechanical computation, and I take "the orthodox view" to be that algorithms can be faithfully represented by these mechanical procedures.[10] The theory developed in [22, 23, 24] starts from a radically different perspective which takes *recursion* rather than *step-by-step computation* as the essence of an

[10] For example, the *computational methods* of Knuth [16] are essentially the same as "finitary" iterators. Knuth reserves the term "algorithm" for computational methods which produce a value on each input, so that most programs do not express algorithms until we have proved that they, in fact converge. It would seem that whatever name we choose for it, the basic notion is that of an effective procedure which might or might not converge.

algorithm. We will review it briefly and discuss just two arguments in its favor, those most relevant to the purpose at hand.

3.2. The meaning of a recursive definition. A typical *mutual* or *simultaneous* recursive definition has the form

$$\left.\begin{array}{rcl} p_1(u_1) & \simeq & f_1(u_1, p_1, \ldots, p_n, x), \\ & \cdots & \\ p_n(u_n) & \simeq & f_n(u_n, p_1, \ldots, p_n, x), \end{array}\right\} \tag{50}$$

where $f_1, \ldots, f_n$ are (monotone) functionals on the individual arguments $u_1, \ldots, u_n, x$ and the partial function arguments $p_1, \ldots, p_n$ and we understand the definition to determine for each value x of *the parameter* the simultaneous *least fixed points* $\overline{p}_1^x, \ldots, \overline{p}_n^x$, the least partial functions which satisfy its n equations in (50) with the fixed x. We may choose to call the first equation *principal* and say that the recursion defines the partial function $\lambda(u_1, x)\overline{p}_1^x(u_1)$, or (more conveniently, following the syntax of LPCR) say that the recursion defines the partial function

$$\overline{\mathbf{f}}(x) \simeq f_0(\overline{p}_1^x, \ldots, \overline{p}_n^x, x), \tag{51}$$

where f_0 is one more functional associated with the recursion, its *head*.

It follows from early results of Kleene in the theory of recursion, that if we have for each $i = 0, \ldots, n$ a Turing machine $\mathcal{T}_i$ which computes $f_i(u_i, p_1, \ldots, p_n)$ (with a suitable coding of the argument u_i and access to "oracles" that yield on demand requested values for each of the p_j's), then we can construct a new Turing machine $\overline{\mathcal{T}}$ which computes $\overline{f}$. The argument generalizes easily to arbitrary iterators. Thus, it is argued, since *recursion can be reduced to iteration*, the notion of iterator is rich enough to represent algorithms expressed by recursive definitions.

A serious problem with this analysis is that the claimed reduction of recursion to iteration is by no means unique or natural. If we apply any of the popular known reductions to the specific, recursive definition of the *mergesort* mentioned above, we get a Turing machine which spends most of its time building "stacks," implementing "calls" and copying parts of its tape, i.e., implementing recursion rather than sorting; and if we use another notion of computational model like the assembly language of some actual computing machine (especially one capable of substantial parallel computation), then we would likely use a different reduction procedure and execution of the resulting program will bear little discernible resemblance to the computation of the Turing machine allegedly expressing the same algorithm. At the same time, the most important properties of recursive algorithms are typically proved directly from the recursive equations which define them, without any reference to details of the specific method by which we aim to reduce the recursion to iteration in order to implement it on a mechanical device.[11]

[11] This is discussed in some detail in [22] for the mergesort, whose basic property is that it will alphabetize a list of n words "using" no more than $n \cdot \log(n)$ comparisons. The result follows almost trivially from the recursive definition of the algorithm and has nothing to do with its many and diverse implementations.

On this account, it seems natural to look for a more abstract notion of *recursive algorithm* which can be read directly from the equations (50), (51), just as its fundamental, implementation-free properties can be proved directly from (50), (51).

3.3. A **recursor** on a set X to W is any tuple of (partial, monotone) functionals $[f_0, f_1, \ldots, f_n]$ whose types are such that the equations (50), (51) make sense when $x \in X$, and where the image of f_0 is W. We write

$$\mathbf{f} = [f_0, f_1, \ldots, f_n] : X \hookrightarrow W, \tag{52}$$

we call $f_0, \ldots, f_n$ *the parts* of $\mathbf{f}$ and we say that $\mathbf{f}$ *computes* the functional $\overline{\mathbf{f}} : X \rightharpoonup W$ defined by (51), its *denotation*. A functional f may be identified with the trivial recursor $[f]$ which computes it. Two recursors $\mathbf{f}$ and $\mathbf{g}$ are *equal* if they have the same number of parts n and for some permutation σ of $\{0, \ldots, n\}$ with $\sigma(0) = 0$ and inverse ρ,

$$f_i(u_i, p_1, \ldots, p_n, x) \simeq g_{\rho(i)}(u_i, p_{\sigma(1)}, \ldots, p_{\sigma(n)}, x), \quad (i = 0, \ldots, n). \tag{53}$$

The definition of recursor identity seems complex, but it simply identifies recursors which determine the same recursive definitions, except for the order in which the equations are listed. For example, if $\mathbf{f} = [f_0, f_1, f_2]$ has three parts and we set

$$\begin{aligned} g_0(p_2, p_1, x) &\simeq f_0(p_1, p_2, x), \\ g_1(u_1, p_2, p_1, x) &\simeq f_1(u_1, p_1, p_2, x), \\ g_2(u_2, p_2, p_1, x) &\simeq f_2(u_2, p_1, p_2, x), \end{aligned}$$

then $\mathbf{g} = [g_0, g_2, g_1] = [f_0, f_1, f_2] = \mathbf{f}$, because $\mathbf{g}$ determines the recursion

$$\begin{aligned} p_2(u_2) &\simeq g_2(u_2, p_2, p_1, x) \simeq f_2(u_2, p_1, p_2, x), \\ p_1(u_1) &\simeq g_1(u_1, p_2, p_1, x) \simeq f_1(u_1, p_1, p_2, x), \\ \overline{g}(x) &\simeq g_0(\overline{p}_2^x, \overline{p}_1^x, x) \simeq f_0(\overline{p}_1^x, \overline{p}_2^x, x), \end{aligned}$$

which is exactly that determined by $\mathbf{f}$, with the first two equations interchanged.

The basic presupposition of the theory of recursive algorithms is that the fundamental mathematical and implementation-free properties of an algorithm can be coded by a recursor: put another way, to define an algorithm which computes a functional $\overline{f}$, it is enough to specify a recursive definition whose head least fixed point is $\overline{f}$. Notice that recursors, like iterators, need not be effective in any way. To separate the *iterative algorithms* from the iterators we need to add a restriction of effectivity, e.g., by insisting that all the objects of the iterator are finitely described, as in the case of Turing machines. In the theory of recursive algorithms we can only talk about *relatively effective algorithms*, as follows.

3.4. Intensional semantics. Fix a functional structure $\mathbf{A}$ of signature τ as in **2.6**. The theory of recursive algorithms associates with each term ϕ of $\mathrm{FLR}(\tau)$ and

each sequence of variables $\vec{\mathrm{x}}$ which includes all the free variables of ϕ, a recursor

$$int_{\mathbf{A}}(\vec{\mathrm{x}}, \phi) : X \hookrightarrow W,$$

where X is the space of all n-tuples $(x_1, \ldots, x_n)$ such that the type of x_i matches that of the formal variable x_i and W is the basic set with type that of the term ϕ. This *referential intension* $int_{\mathbf{A}}(\vec{\mathrm{x}}, \phi)$ of ϕ in $\mathbf{A}$ computes the (functional) denotation $den_{\mathbf{A}}(\vec{\mathrm{x}}, \phi) : X \rightharpoonup W$ of ϕ in $\mathbf{A}$. The *recursive algorithms* of $\mathbf{A}$ are the recursors which occur as intensions of FLR-terms on $\mathbf{A}$, and the *recursive functionals* of $\mathbf{A}$ are their denotations.

The precise definition of intensions given in [24] is a bit complex,[12] it depends on several technical results of [23] and we will not attempt to reproduce it here. The main task is to explain how recursive definitions can be combined directly, without the introduction of extraneous, implementation dependent data structures, i.e., to interpret the formulas (50), (51) when $f_0, \ldots, f_n$ are recursors. The examples of referential intensions below will illustrate how the notions work.

3.5. Ideal, infinitary recursive algorithms. Consider the recursive definition of the partial truth predicate $T_{\mathbf{A}}$ for the language of LPCR on an acceptable structure $\mathbf{A}$ as in (21). It can be written out in the form (50), (51), but two of the clauses will involve the "infinitary" operations $\forall_A$, $\exists_A$. This means that in the associated recursor

$$\mathbf{T_A} = [T_0, T_1, \ldots, T_n] : A \hookrightarrow TV \tag{54}$$

two of the functionals are of the form

$$\begin{array}{rcl} T_i(u, P_1, \ldots, P_n) & \simeq & \forall_A(\lambda(y)P_k(u, y)), \\ T_j(u, P_1, \ldots, P_n) & \simeq & \exists_A(\lambda(y)P_k(u, y)). \end{array} \tag{55}$$

On the account we are giving, $\mathbf{T_A}$ is a recursive algorithm of $\mathbf{A}$ which computes the partial truth predicate $T_{\mathbf{A}}$. There is little doubt that on its own, $\mathbf{T_A}$ is an interesting mathematical object, it is the semantic version of the definition of truth. On the other hand, as an "algorithm," it cannot be implemented when $\mathbf{A}$ is infinite, however we define "implementations." So there is a real question whether it is useful to adopt the terminology of algorithms in connection with such infinitary objects: do our intuitions about algorithms garnered from constructing and analyzing finitary, step-by-step mechanical procedures carry over to help us understand and study such infinitary objects?

One point to consider is the standard and highly developed *fixed point theory of programs* which (in effect) interprets ordinary programs by recursive equations. Almost all the concepts and methods of this theory extend to arbitrary recursive algorithms, sometimes trivially, often with some extra effort. Recursion in higher types [14, 15], [25], [13], recursion on admissible ordinals and set recursion [27], positive elementary induction [21] and other *generalized recursion theories*

[12] In fact we have oversimplified a bit, because only the so-called *special terms* define algorithms of a structure. The subtleties about special terms are not relevant to the modeling of Frege's sense by the referential intension and we will disregard them in this paper.

[11, 10] have been developed and found fruitful applications in many parts of logic, set theory and computer science based on this idea. The results in this subject are typically stated in the form of extensional properties of various collections of "computable" or "recursive" functions, but the proofs always depend on intensional properties of the recursions by which these functions are defined. For example, Kripke [19] states at the end of his paper that (on an acceptable structure) the relations weakly definable in his language are exactly the inductive relations, and those strongly definable (by formulas with no truth gaps) are the hyperelementary relations. This is an extensional statement, but its proof depends on the *Stage Comparison Theorem* 2A.2 of [21], which is most naturally understood as a statement about the lengths of the "abstract computations" by the recursor $\mathbf{T_A}$.[13]

The intension of the *liar* $\equiv P(\,)$ where $\{P(\,) \simeq \neg P(\,)\}$ is the recursor

$$\mathrm{l} = [l_0, l_1], \text{ where } l_0(P) \simeq P(\,),\ l_1(P) \simeq \neg P(\,), \tag{56}$$

and that of the *truthteller* $\equiv P(\,)$ where $\{P(\,) \simeq P(\,)\}$ is

$$\mathbf{tt} = [l_0, l_0]. \tag{57}$$

We can read these directly from the sentences involved, because they are so simple. In general we need to *reduce* a sentence by the *reduction calculus* of [23] which preserves intensions, until we bring it to an irreducible *normal form*, from which we can read off its intension.

Suppose R, Q are interpreted by converse relations R, Q in a structure and a, b are constants interpreted by elements a, b. The reduction calculus gives the normal forms

$$\begin{aligned} \mathrm{R(a,b)} &\sim_{\mathbf{A}} \mathrm{R}(p_1(\,), p_2(\,)) \text{ where } \{p_1(\,) \simeq \mathrm{a},\ p_2(\,) \simeq \mathrm{b}\}, \\ \mathrm{Q(b,a)} &\sim_{\mathbf{A}} \mathrm{Q}(p_2(\,), p_1(\,)) \text{ where } \{p_2(\,) \simeq \mathrm{b},\ p_1(\,) \simeq \mathrm{a}\}, \end{aligned}$$

and the three-part intensions that we can read off these normal forms are equal (via the permutation $\sigma(1) = 2$, $\sigma(2) = 1$) precisely because R and Q are converse relations. A similar computation shows that in general,

$$\phi \,\&\, \psi \sim_{\mathbf{A}} \psi \,\&\, \phi, \qquad \phi \vee \psi \sim_{\mathbf{A}} \psi \vee \phi.$$

As an additional example from LPC, consider the sentence χ of (4). We exhibit some of the steps of its reduction to normal form, to illustrate the reduction calculus of [23]. The six-part intension of χ which can be read off this normal form is the recursor representation of the algorithm described by Steps (1)–(4) in Section 1.

[13] Kripke also associates with each sentence ϕ which has a truth value its *level*, the least ordinal at which his truth definition yields that value. In algorithmic terms, this is precisely the length of the computation by the recursor $\mathbf{T_A}$ on the Gödel code of ϕ. A more natural level function assigns to each ϕ the length of the computation of the recursor $int(\phi)$, with no appeal to Gödel codes.

$$
\begin{aligned}
\chi &\equiv (\forall \mathrm{x})(\exists \mathrm{y})\mathrm{R}(\mathrm{x},\mathrm{y}) \vee (\exists \mathrm{z})\mathrm{Q}(\mathrm{z})\\
&\sim_{\mathbf{A}} \lambda(\,)P_1(\,) \vee \lambda(\,)P_2(\,)\\
&\quad \textsf{where } \{P_1(\,) \simeq (\forall \mathrm{x})(\exists \mathrm{y})\mathrm{R}(\mathrm{x},\mathrm{y}),\\
&\qquad\quad P_2(\,) \simeq (\exists \mathrm{z})\mathrm{Q}(\mathrm{z})\}\\
&\sim_{\mathbf{A}} \lambda(\,)P_1(\,) \vee \lambda(\,)P_2(\,)\\
&\quad \textsf{where } \{P_1(\,) \simeq \forall_A(P_{11}) \textsf{ where } \{P_{11}(\mathrm{x}) \simeq (\exists \mathrm{y})\mathrm{R}(\mathrm{x},\mathrm{y})\},\\
&\qquad\quad P_2(\,) \simeq \exists_A(P_{22}) \textsf{ where } \{P_{22}(\mathrm{z}) \simeq \mathrm{Q}(\mathrm{z})\}\\
&\qquad\quad \}\\
&\sim_{\mathbf{A}} \lambda(\,)P_1(\,) \vee \lambda(\,)P_2(\,)\\
&\quad \textsf{where } \{P_1(\,) \simeq \forall_A(P_{11}),\ P_{11}(\mathrm{x}) \simeq (\exists \mathrm{y})\mathrm{R}(\mathrm{x},\mathrm{y}),\\
&\qquad\quad P_2(\,) \simeq \exists_A(P_{22}),\ P_{22}(\mathrm{z}) \simeq \mathrm{Q}(\mathrm{z})\\
&\qquad\quad \}\\
&\sim_{\mathbf{A}} \ \ldots\\
&\sim_{\mathbf{A}} \lambda(\,)P_1(\,) \vee \lambda(\,)P_2(\,)\\
&\quad \textsf{where } \{P_1(\,) \simeq \forall_A(P_{11}),\ P_{11}(\mathrm{x}) \simeq \exists_A(\lambda(\mathrm{y})P_{111}(\mathrm{x},\mathrm{y})),\\
&\qquad\quad P_{111}(\mathrm{x},\mathrm{y}) \simeq \mathrm{R}(\mathrm{x},\mathrm{y}),\ P_2(\,) \simeq \exists_A(P_{22}),\ P_{22}(\mathrm{z}) \simeq \mathrm{Q}(\mathrm{z})\\
&\qquad\quad \}.
\end{aligned}
$$

Finally, the "bald King of France" sentence (36) has the following normal form, assuming that both $KF(x)$ and $B(x)$ are prime formulas:

$$BKF \sim_{\mathbf{A}} P(q(\,)) \textsf{ where } \{P(x) \simeq B(x),\ q(\,) \simeq \textsf{the } (R),\ R(x) \simeq KF(x)\}.$$

Its intension has four parts, the head being

$$bkf_0(P,q,R) \simeq P(q(\,)).$$

This and the preceding two examples illustrate the simple form of the intensions of sentences in LPC, even when we extend it with description operators or (quite easily) generalized connectives and quantifiers, modal operators and the like. They describe recursions which, however non-constructive (when the functionals in the structure are non-constructive), they are trivial as recursions, they "close" in a finite number of steps. Except for the imagery of algorithms, the theory on this level is really not that much different from Church's (Alternative 0 version) in [1] or Cresswell's, which could cover the same ground with the introduction of a natural equivalence relation on proposition surrogates or structures. It is, of course, the algorithm imagery which justifies the transformations by which we reduce sentences to normal form, compute their intensions and then use the natural identity of these recursors to get precisely the equivalence relation which is needed. When we add self-reference to the language we obtain the less trivial intensions of formulas such as the truth predicate, which describe more interesting recursions. We also make considerably more complex the relation of intensional identity.

3.6. THEOREM. *The problem of graph isomorphism for finite (unordered) graphs is polynomially reducible to the problem of intensional identity, for sentences of* LPCR *on any structure.*

PROOF (Kit Fine).[14] Given a graph with n nodes $P_1, \ldots, P_n$ and $k \leq n^2$ edges $E_1, \ldots, E_k$, we construct a sentence $\phi(G)$ using $P_1, \ldots, P_n, E_1, \ldots, E_k$ and an additional Q as partial propositional variables. The sentence $\phi(G)$ is a **where** sentence with $n + k + 1$ parts, the head being $Q(\,)$ and the other parts of the form

$$\begin{aligned} P_i(\,) &\simeq P_i(\,), \quad (i = 1, \ldots, n), \\ E_j(\,) &\simeq P_{b(j)}(\,) \vee P_{e(j)}(\,), \quad (j = 1, \ldots, k), \end{aligned}$$

where for each j, $P_{b(j)}$ and $P_{e(j)}$ are respectively the beginning and the end nodes of the edge E_j. The construction is obviously polynomial and it is quite trivial to verify that two graphs are isomorphic exactly when the corresponding sentences have the same intension, directly from the definition of recursor identity. $\dashv$

All the sentences constructed in the proof receive no truth value, they are silly and complex versions of the truthteller. One can build more interesting examples of complex intensions, but this simple construction exhibits the complexity of deciding intensional identity, the graph isomorphism problem being notoriously difficult.

3.7. A **recursor structure** of signature $\tau = (B, \{\mathrm{f}_1, \ldots, \mathrm{f}_n\}, d)$ is an interpretation of FLR(τ) where the function symbols are interpreted by recursors, i.e., a tuple $\mathbf{A} = (\mathcal{U}, \mathbf{f}_1, \ldots, \mathbf{f}_n)$, where each $\mathbf{f}_i$ is a recursor with head fixed point a functional $\overline{\mathbf{f}}_i$ of type $d(\mathrm{f}_i)$. We can think of every functional structure as a recursor structure, identifying its functionals with the associated trivial recursors which compute them, as in **3.3**. Recursor structures arise naturally when we enrich a language by adding a new symbol to express a complex notion, and then we want the new symbol to be interpreted by the complex meaning of the expression it replaces. Kripke's language is naturally interpreted in recursor structures, since (at the least) we want the ordinals attached to sentences to be part of the interpretation of T. The theory of intensions generalizes directly to the category of recursor structures, which is the natural context for it.

3.8. Non-minimal fixed points. In [19], Kripke discusses several possibilities of interpreting self-referential sentences by choosing fixed points other than the least one. He says that "the smallest fixed point is probably the most natural model for the intuitive concept of truth," and the algorithmic analogy supports this. On the other hand, the referential intension of a sentence determines not only the smallest fixed point but all of them; thus, if we identify *sense* with *referential intension*, we do not lose those aspects of meaning which may be represented by the non-minimal fixed points. For example, Kripke calls the *liar* "paradoxical" because it cannot have a truth value in any fixed point of $\mathbf{T}_\mathbf{A}$. The paradoxical nature of the *liar* is coded by its referential intension, a recursor with the property that none of its simultaneous fixed points yields a truth value.

[14] This argument of Fine's (included with his permission) is simpler than my original proof and shows that in fact *the problem of intensional identity for the propositional part of* LPCR is at least as hard as the graph isomorphism problem.

§4 Sense identity and indirect reference.

Van Heijenoort [29] quotes an extensive passage from a 1906 letter from Frege to Husserl which begins with the following sentence:

> "It seems to me that we must have an objective criterion for recognizing a thought as the same thought, since without such a criterion a logical analysis is not possible."

This could be read as asserting the existence of a decision procedure for sense identity, but unfortunately, the letter goes on to suggest that logically equivalent sentences have the same sense, a position which is contrary to the whole spirit of [12]. It is apparently not clear what Frege thought of this question or if he seriously considered it at all. Kreisel and Takeuti [17] raise explicitly the question of *synonymity* of sentences which may be the same as that of identity of sense. If we identify sense with referential intension, the matter is happily settled by a theorem.

4.1. THEOREM. *For each recursor structure* $\mathbf{A} = (U_1, \ldots, U_k, f_1, \ldots, f_n)$ *of finite signature, the relation* $\sim_{\mathbf{A}}$ *of intensional identity on the terms of* FLR *interpreted on* $\mathbf{A}$ *is decidable.*

For each structure $\mathbf{A}$ and arbitrary integers n, m, let

$$S_{\mathbf{A}}(n,m) \iff n,\ m \text{ are Gödel numbers of sentences or terms } \theta_n,\ \theta_m \text{ of FLR and } \theta_n \sim_{\mathbf{A}} \theta_m. \tag{58}$$

The rigorous meaning of **4.1** is that this relation $S_{\mathbf{A}}$ is decidable, i.e., computable by a Turing machine. By the usual coding methods then, we get immediately:

4.2. COROLLARY. *The relation* $S_{\mathbf{A}}$ *of intensional identity on Gödel numbers of expressions of* FLR *is elementary (definable in* LPC*), over each acceptable structure* $\mathbf{A}$.

The Corollary is useful because it makes it possible to talk indirectly about FLR intensions within FLR. In general, we cannot do this directly because the intensions of a structure $\mathbf{A}$ are higher type objects over $\mathbf{A}$ which are not ordinarily[15] members of any basic set of the universe of $\mathbf{A}$. One reason we might want to discuss FLR intensions within FLR is to express *indirect reference*, where Frege's treatment deviates from his general doctrine of separate compositionality principles for sense and denotation. Frege argued that "the indirect denotation of a word is ... its customary sense," so that in

Othello believed that Cassio and Desdemona were lovers, (59)

[15]If the universe of $\mathbf{A}$ contains the powerset of every basic set in it and the Cartesian product of every two basic sets, then of course it contains all recursors over basic sets and with suitably rich primitives we can develop the theory of intensions of $\mathbf{A}$ within LPCR. These are typed structures, however, of infinite signature, which lack a natural universe, a largest basic set. More interesting would be the structure of the universe of sets, whose only basic "set" is the class of all sets. The intensions of this structure are certainly not sets.

the object of Othello's belief would be the sense of the sentence 'Cassio and Desdemona were lovers'. Since we cannot refer to sense directly in any fragment of natural language formalizable within FLR (if we identify it with intension), we might attempt to make belief an attribute of sentences, or (equivalently) their Gödel numbers. This means that (59) is expressed by

$$\text{Othello believed 'Cassio and Desdemona were lovers'}, \tag{60}$$

where 'Cassio and Desdemona were lovers' is the Gödel number of the sentence within the quotes. But then we would certainly expect (60) to imply

$$\text{Othello believed 'Desdemona and Cassio were lovers'}, \tag{61}$$

and we would like to express in the language the general assertion that belief depends only on the sense, not the syntactic form of a sentence, i.e.,

$$[\text{Othello believes } m \ \&\ S_{\mathbf{A}}(m,n)] \Longrightarrow \text{ Othello believes } n. \tag{62}$$

The Corollary says precisely that (62) is expressible already in LPC. The method is evidently quite general: if we view propositional attitudes as attributes of Gödel codes, we can express in the language that they respect sense identity, those indeed which should respect it.

PROOF OF THE MAIN THEOREM **4.1**. The intension of a term in a recursor structure is computed in the associated functional expansion by Def. **3.6** of [24], so we may assume that the interpretations $f_1, \ldots, f_n$ of the function symbols of the language in $\mathbf{A}$ are functionals. We fix such a functional structure $\mathbf{A}$ then and we assume (for the time being) that *all basic sets in* $\mathbf{A}$ *are infinite*. We will discuss the interesting case of finite basic sets at the end.

Since intensions are preserved by passing to the normal form, the problem of intensional identity on $\mathbf{A}$ comes down to this: given two irreducible, recursive terms

$$\begin{aligned} \phi &\equiv \phi_0 \text{ where } \{p_1(u_1) \simeq \phi_1, \ldots,\ p_n(u_n) \simeq \phi_n\}, \\ \psi &\equiv \psi_0 \text{ where } \{q_1(v_1) \simeq \psi_1, \ldots,\ q_m(v_m) \simeq \psi_m\}, \end{aligned}$$

is $n = m$ and can we match the terms so that they define the same functionals? By trying all possible ways to match the parts[16] and using the form of irreducible, explicit terms (2B.4 of [23]), we can further reduce the problem to that of deciding whether an arbitrary identity in one of the following three forms holds in $\mathbf{A}$:

$$f(z_1, \ldots, z_m) \simeq g(z_{m+1}, \ldots, z_l), \tag{63}$$

$$f(z_1, \ldots, z_m) \simeq p(w_1, \ldots, w_k). \tag{64}$$

$$q(w_1, \ldots, w_m) \simeq p(w_{m+1}, \ldots, w_l). \tag{65}$$

Here the following conditions hold:

1. The functionals f and g are among the finitely many givens of $\mathbf{A}$, or the constants $\mathbf{t}$, $\mathbf{f}$ or the conditional.

[16] This trivial part of the algorithm is (on the face of it) in NP (non-deterministic, polynomial time) in the length of the given terms and the rest will be seen to be no worse. I do not know a better upper bound for the complexity of intensional identity on a fixed structure and the best lower bound I know is that of Theorem **3.6**.

2. Each z_i is an *immediate expression* (in the full set of variables) by 2B.2 of [23], i.e., either a basic variable, or $p(\vec{x})$ where the x_i's are basic variables, or $\lambda(\vec{s})p(\vec{x})$ with $p(\vec{x})$ as above.

3. Each w_j is either a basic variable or $r(\vec{x})$, where the x_i's are basic variables and where $r \equiv p$ and $r \equiv q$ are allowed.

The decision question is trivial for identities in form (65) because of the following elementary result from equational logic.

4.3. LEMMA. *An identity* (65) *is valid on any fixed structure with infinite basic sets only if its two sides are identical.*

We can view (64) as a special case of (63), with the following *evaluation functional* substituted for g on the right:

$$ev^k(p, x_1, \ldots, x_k) \simeq p(x_1, \ldots, x_k). \tag{66}$$

Notice, however, that there are infinitely many such evaluation functionals. There are also infinitely many possible identities in form (63), because although f and g are chosen from a finite set, there is an infinite number of immediate expressions from which to choose the z_i's. The proof splits into two parts. First we will show that if we expand the structure by a fixed, finite number of evaluation functionals, then every identity in form (64) is effectively equivalent to one in form (63). In the second part we will show how to decide the validity of equations in form (63).

4.4. A basic variable v is *placed* in an identity (63) or (64) if $v \equiv z_i$ for some i. For example, the placed variables of $f(v, p(x, y), u) = p(s, r(x, y))$ are v and u.

4.5. LEMMA.[17] *Suppose the identity*

$$f(z_1, \ldots, z_m) \simeq p(w_1, \ldots, w_k). \tag{67}$$

is valid in the structure $\mathbf{A}$ *with infinite basic sets and* $w_i \equiv r(\vec{x})$ *is one of the terms on the right. Then there exists some* z_j *on the left such that either* $w_i \equiv z_j$, *or* $z_j \equiv \lambda(s_1, \ldots, s_k)r(\vec{y})$ *and* $r(\vec{x})$ *can be obtained from* $r(\vec{y})$ *by the substitution of placed variables for* $s_1, \ldots, s_k$.

PROOF. To keep the notation simple we assume that there is only one basic set and we consider the special case where

$$w_2 \equiv r(x, u, x, y, v), \quad u,\ v \text{ placed},\ x,\ y \text{ not placed}. \tag{68}$$

CASE 1. $r \not\equiv p$. Choose disjoint sets D_{x}, D_{y}, D_{u}, D_{v}, W and some $\bar{c}$ outside all of them and first set all variables other than x, y, u, v to $\bar{c}$ and all partial function variables other than r, p to constant functions with value $\bar{c}$. Next set u and v to constant values $\bar{u}$, $\bar{v}$ in the corresponding sets D_{u}, D_{v}. For each arbitrary partial function

$$\omega : D_{\mathrm{x}} \times D_{\mathrm{x}} \times D_{\mathrm{y}} \rightharpoonup W,$$

[17] I am grateful to Joan Moschovakis for a counterexample which killed a plausible simplification of this proof, before I invested too much time in it.

set r by the conditions

$$[s_1 \notin D_{\mathrm{x}} \vee s_2 \notin D_{\mathrm{u}} \vee s_3 \notin D_{\mathrm{x}} \vee s_4 \notin D_{\mathrm{y}} \vee s_5 \notin D_{\mathrm{v}}] \Longrightarrow r(s_1, s_2, s_3, s_4, s_5) \simeq \bar{c},$$

$$[s_1 \in D_{\mathrm{x}} \;\&\; s_2 \in D_{\mathrm{u}} \;\&\; s_3 \in D_{\mathrm{x}} \;\&\; s_4 \in D_{\mathrm{y}} \;\&\; s_5 \in D_{\mathrm{v}}] \Longrightarrow r(s_1, s_2, s_3, s_4, s_5) \simeq \omega(s_1, s_3, s_4)$$

and finally set

$$\sigma(t) = \begin{cases} t, \text{ if } t \in W, \\ \bar{c}, \text{ otherwise,} \end{cases} \qquad p(s_1, s_2, \ldots, s_k) \simeq \sigma(s_2). \tag{69}$$

Consider the result of further substituting in (67) arbitrary values $x \in D_{\mathrm{x}}$, $y \in D_{\mathrm{y}}$. Suppose $w_j \equiv q(\vec{t})$ is one of the terms within p on the right. If $q \not\equiv r$, then with these substitutions w_j is defined, set either to $\bar{c}$ or to $\sigma(t_2)$, if $q \equiv p$. If $q \equiv r$ and the sequence of variables $\vec{t}$ is not *exactly* x, u, x, y, v, then w_j again takes the value $\bar{c}$. Thus the only term which may possibly be undefined among the w_j's is w_2 (which may of course occur more than once) and hence the right-hand-side of (67) is defined exactly when w_2 is defined and we have a valid identity:

$$f(z_1(\omega, x, y), \ldots, z_m(\omega, x, y)) \simeq \omega(x, x, y), \tag{70}$$
$$(x \in D_{\mathrm{x}},\ y \in D_{\mathrm{y}},\ \omega : D_{\mathrm{x}} \times D_{\mathrm{x}} \times D_{\mathrm{y}} \rightharpoonup W).$$

The typical expression $z_i(\omega, x, y)$ on the left evaluates to the constant $\bar{c}$ or some function with the constant value $\bar{c}$ if neither r nor p occurs in z_i. If $z_i \equiv \lambda(\vec{s})p(\vec{t})$, then again z_i has a value independent of ω, x, y, because of the definition of p and the fact we we set no variable equal to a member of W. Finally, if

$$z_i \equiv \lambda(\bar{s})r(t_1, t_2, t_3, t_4, t_5), \tag{71}$$

but some t_i is free or a constant and is not the ith variable or constant in the pattern $x, \bar{u}, x, y, \bar{v}$, then again the expression evaluates to $\bar{c}$, by the definition of r. Thus z_i depends on ω, x, y only when at most t_1, t_3 or t_4 are free, and those among them which are free are set to the corresponding value x, x or y. If all three are free in some such z_i, then the lemma clearly holds. In the opposite case the partial function ω satisfies an identity of the form

$$\omega(x, x, y) \simeq h(\omega, \omega(\cdot, x, y), \omega(x, \cdot, y), \omega(x, x, \cdot), \omega(\cdot, \cdot, y), \omega(\cdot, x, \cdot), \omega(x, x, \cdot)), \tag{72}$$

where h is a monotone operation on partial functions and $\cdot$ is the algebraic notation for λ-abstraction, e.g.,

$$\omega(\cdot, x, \cdot) = \lambda(s, t)\omega(s, x, t).$$

For example, suppose

$$z_i \equiv \lambda(st)r(x, \bar{u}, s, t, \bar{v}) = \beta;$$

then

$$\beta(s, t) \simeq \begin{cases} \omega(x, s, t), \text{ if } s \in D_{\mathrm{x}},\ t \in D_{\mathrm{y}}, \\ \bar{c}, \text{ otherwise,} \end{cases}$$

so that $z_i = h_i(\omega(x, \cdot, \cdot))$ with a monotone h_i. A similar evaluation of z_i in terms of some *section* of ω is involved in each of the cases and the substitution of all these monotone h_i's into f yields a monotone operation.

Finally, we obtain a contradiction from the alleged validity of (72). Choose distinct points x_0, x_1, y_0, y_1 in the respective sets $D_{\rm x}$, $D_{\rm y}$ and define two partial functions with only the indicated values, where $0, 1$ are distinct points in W.

$$\alpha(x_0, x_0, y_0) \simeq 0, \quad \gamma(x, x', y) \simeq \begin{cases} 0, \text{ if } x = x_0 \vee x' = x_0 \vee y = y_0, \\ 1, \text{ otherwise.} \end{cases}$$

¿From (72) applied to α,

$$h(\alpha, \alpha(\cdot, x_0, y_0), \ \ldots, \ ) \simeq \alpha(x_0, x_0, y_0) \simeq 0. \tag{73}$$

But obviously $\alpha \subseteq \gamma$ and an easy computation shows that every section of γ at (x_1, x_1, y_1) extends the corresponding section of α at (x_0, x_0, y_0), for example

$$\lambda(s)\alpha(x_0, s, y_0) \subseteq \lambda(s)\gamma(x_1, s, y_1),$$

simply because $\gamma(x_1, x_0, y_1) \simeq 0$. Thus by the monotonicity of h, (73) and (72) applied to γ, we should have

$$0 \simeq g(\gamma, \gamma(\cdot, x_1, y_1), \ \ldots, \ ) \simeq \gamma(x_1, x_1, y_1),$$

while by its definition $\gamma(x_1, x_1, y_1) \simeq 1$. This completes the proof of the Lemma in the first case.

CASE 2. $r \equiv p$. We consider again a typical, simple case

$$f(z_1, \ldots, z_m) \simeq p(w_1, p(x, y, u), w_2), \ u \text{ placed}, \ x, y \text{ not placed}.$$

As before, we restrict the variables to disjoint sets $D_{\rm x}$, $D_{\rm y}$, $D_{\rm u}$, W and we set:

$$p(s_1, s_2, s_3) \simeq \begin{cases} \omega(s_1, s_2), \text{ if } s_1 \in D_{\rm x}, \ s_2 \in D_{\rm y}, \ s_3 \in D_{\rm u}, \\ s_2, \text{ otherwise, if } s_2 \in W, \\ \bar{c}, \text{ otherwise.} \end{cases}$$

¿From this it follows that we get a valid identity of the form (72) for an arbitrary $\omega : D_{\rm x} \times D_{\rm y} \rightharpoonup W$, the main points being that all the terms on the right which are not identical with w_2 are defined and only the sections show up on the left, and then the proof is finished as before. ⊣

4.6. LEMMA. *An identity of the form*

$$f(z_1, \ldots, z_m) \simeq p(w_1, \ldots, w_k) \tag{74}$$

cannot be valid in a structure $\mathbf{A}$ *with infinite basic sets if the number* n *of distinct terms (not variables) on the right is greater than a fixed number* d*, which depends only on the type of* f*; if* $n \leq d$*, then we can compute from* (74) *an equivalent identity of the form*

$$f(z_1, \ldots, z_m) \simeq ev^n(W_0, W_1, \ldots, W_n). \tag{75}$$

PROOF. If (74) is valid, then by the preceding Lemma **4.5**, each w_i which is a term either is identical with some z_j or can be obtained by the substitution of placed variables in some z_j. If there are $q \leq m$ placed variables, and if z_j is a λ-term, it is of the form $\lambda(s_1, \ldots, s_{l(j)})z_j^*$, where the number $l(j)$ can be computed from the type of f, so it can generate by substitution of placed variables into its

bound variables at most $q^{l(j)}$ distinct terms; hence the total number of distinct terms on the right cannot exceed

$$d = \sum_{j=1}^{m} q^{l(j)}. \tag{76}$$

Suppose the right-hand-side of (74) is $p(x, A, u, B, A, z)$, where distinct caps indicate distinct terms and the lower case letters are variables. We then have

$$\begin{aligned} p(x, A, u, B, A, x) &\simeq (\lambda(a, b)p(x, a, u, b, a, x))(A, B) \\ &\simeq ev^2(\lambda(a, b)p(x, a, u, b, a, x), A, B). \end{aligned}$$

The general case is similar. ⊣

This last lemma reduces the decision problem of intensional identity to equations in form (63), where there is a finite choice of f's, the functionals in the signature, and a finite choice of g's, those in the structure and the ev^k's, for k less than d computed by (76) for every functional in the structure.

4.7. Extended sets and assignments. Before describing the procedure which determines the validity of identities in form (63), we consider a simple example which illustrates one of the annoying subtleties we will need to deal with. Suppose g is a total, unary function on some set A and we define the total, binary function f by

$$f(x, y) \simeq g(x). \tag{77}$$

Clearly (77) is a valid identity involving the old g and the new f we just defined. Suppose we substitute a term in this to get

$$f(x, p(x)) \simeq g(x); \tag{78}$$

now this is not valid, because for some values of the partial function p, $p(x)$ will not be defined, so neither will $f(x, p(x))$, while the right-hand-side is defined. In dealing with partial functions and functionals as we have, *validity of identities is not preserved by substitution of terms.* One way to deal with this problem is to add an element $\perp$ to each basic set and view partial functions as total functions, which take the value $\perp$ when they should be undefined. For each set A, we set

$$A^{\perp} = A \cup \{\perp\} = \text{ the extension of } A. \tag{79}$$

We can now try to interpret identities by allowing the basic variables to range over the extended sets, so that the validity of (77) implies the validity of (78); this is fine, except that now (77) *fails for the f we originally defined*, because we still have $f(x, \perp) = \perp \neq g(x)$, when $x \in A$. Of course, some might say we defined the wrong f, but in fact it is these "strict" identities we need to decide to settle the question of intensional identity. In practice we will need to work both with strict and with extended identities and we must keep the context clear.

We will use "=" to denote equality in the extended basic sets and set

$$x\downarrow \iff x \in A \iff x \neq \perp, \quad (x \in A^{\perp}). \tag{80}$$

A *strict assignment* π in a structure $\mathbf{A}$ assigns partial functions to pf variables and members of the basic sets to basic variables, as usual. An *extended assignment*

behaves exactly like a strict assignment on pf variables, but assigns members of the extended basic sets to the basic variables, i.e., it can set $\pi(v) = \perp$. An identity is *strictly valid* when it holds for all strict assignments, and *extendedly valid* if it holds for all extended assignments.

4.8. Dictionary lines. Choose once and for all fixed, *special variables* $x_1, \ldots, x_l$ of types such that

$$f(x_1, \ldots, x_m) \simeq g(x_{m+1}, \ldots, x_l) \tag{81}$$

is well formed. A *dictionary line* for f and g is an implication of the form

$$\phi_1,\ \phi_2,\ \ldots,\ \phi_n \Longrightarrow f(x_1, \ldots, x_m) \simeq g(x_{m+1}, \ldots, x_l) \tag{82}$$

where each formula ϕ_k in the *antecedent of the line* may involve additional *extra, basic variables* other than the $x_1, \ldots, x_l$ and satisfies one of the following conditions.

1. $\phi_k \equiv x_i = u$, where x_i is one of the special, basic variables. At most one formula of this type in the antecedent involves each x_i.

2. ϕ_k is $\lambda(\vec{s})x_i(\vec{u}) = \lambda(\vec{s})x_j(\vec{v})$ or $x_i(\vec{u}) = x_j(\vec{v})$. At most one formula of this type in the antecedent involves each pair x_i and x_j.

3. ϕ_k is $u\downarrow$ or $u \neq v$, where the basic variables u, v occur free in the line in formulas of type (1) or (2).

4.9. Dictionaries. A line is valid (in the given structure) if every extended assignment which satisfies its hypothesis also satisfies its conclusion. This means that the choice of specific extra variables is irrelevant to the validity of a line, and then a simple counting argument shows that the types of f and g determine an upper bound on the number of distinct (up to alphabetic change and reordering of hypotheses) lines. We fix a sufficiently large set of extra variables and list once and for all, all the lines in these variables which are valid for f and g in the given structure; this is *the dictionary for f and g.*

The dictionary of the structure **A** is the union of the dictionaries for all the pairs of functionals in **A**. It is a finite list of lines, perhaps not easy to construct for specific structures with non-constructive givens, but in principle it can be written down.

We will associate (effectively) with each identity (63) a specific set of lines L such that the *strict validity* of (63) is equivalent to the *extended validity* of all the lines in L. It will be convenient to express these lines using the variables which occur in (63). To decide a specific (63), we translate the lines of L into equivalent lines in the fixed, chosen variables by an alphabetic change, and then (63) will be equivalent to the presence of these lines in the dictionary.

For example, (77) will be expressed (essentially) by the single line

$$x_1 = x_3,\ x_1\downarrow,\ x_2\downarrow,\ x_3\downarrow \Longrightarrow f(x_1, x_2) = g(x_3),$$

which is valid, while for (78) we will get

$$x_1 = x_3,\ x_1\!\downarrow,\ x_3\!\downarrow \Longrightarrow f(x_1, x_2) = g(x_3),$$

which is not. (Actually there will be some additional "fringe" on the lines produced by the formal decision procedure, which will not affect the validity of the lines.)

4.10. Bound variable unifiers. Let

$$E \equiv \lambda(u_1, \ldots, u_m)A, \quad F \equiv \lambda(v_1, \ldots, v_n)B, \tag{83}$$

be two immediate λ-expressions. A *bound variable unifier* or just *bvu* for these two expressions is a triple

$$(\tau, \sigma, \vec{s}) = (\tau, \sigma, (s_1, \ldots, s_l)),$$

where

$$\tau : \{u_1, \ldots, u_m\} \to \text{ variables}, \quad \sigma : \{v_1, \ldots, v_n\} \to \text{ variables}$$

are substitution maps on the bound variables, $s_1, \ldots, s_l$ are variables which do not occur in A, B but do occur in both $\tau[A]$ and $\sigma[B]$, and

$$\lambda(s_1, \ldots, s_l)\tau[A] \equiv \lambda(s_1, \ldots, s_l)\sigma[B]. \tag{84}$$

The variables $s_1, \ldots, s_l$ are the *bound variables* of the unifier.

It will be convenient to assume that such variable transformations are defined on all variables, by setting

$$\tau(w) \equiv w, \text{ if } w \text{ is not in the domain of } \tau.$$

We have already used this convention in writing $\tau[A]$, presumably meaning the term resulting by replacing every basic variable u in A by $\tau(u)$.

For example, we can unify

$$\lambda(u, u')r(u, u', a), \quad \lambda(v)r(v, b, a)$$

by setting

$$\tau(u) \equiv \tau(u') \equiv b, \quad \sigma(v) \equiv b, \quad \vec{s} = \emptyset,$$

which identifies both expressions with $\lambda(\,)r(b, b, a)$. It is obvious that this is not the best we can do, though, since we can also set

$$\tau'(u) \equiv s,\ \tau'(u') \equiv b, \quad \sigma'(v) \equiv s, \quad \vec{s} = (s),$$

which unifies the terms "further" to $\lambda(s)r(s, b, a)$. The next definition and lemma capture this simple idea of the existence of a unique such "maximal" unifier, when one exists at all.

4.11. Suppose $(\tau, \sigma, \vec{s})$ and $(\tau', \sigma', \vec{s}')$ are both bvu's for two λ-terms E and F. We say that $(\tau, \sigma, \vec{s})$ is *reducible to* $(\tau', \sigma', \vec{s}')$ if it "unifies no more," i.e., every variable in the sequence $\vec{s}$ is also in the sequence $\vec{s}'$, if $\tau'(w)$ is not bound in $(\tau', \sigma', \vec{s}')$ then $\tau(w) = \tau'(w)$, if $\tau(w)$ is bound in $(\tau, \sigma, \vec{s})$ then $\tau(w) = \tau'(w)$, and similarly with σ, σ'.

A bvu for two λ-terms is *maximal*, if there exists no other bvu for the same λ-terms with a longer sequence of bound variables.

4.12. LEMMA. *(1) Every bounded variable unifier for two λ-terms as in* (83) *can be reduced to a maximal one.*

(2) Two λ-terms have at most one maximal unifier, up to alphabetic change of the bound variables.

PROOF. The *critical number* for two λ-terms as in (83) is the number of distinct variables among the $u_1, \ldots, u_m$, $v_1, \ldots, v_n$ which actually occur in A and B. We will show by induction on the critical number, simultaneously, that if a bvu exists, then a unique maximal one exists and the given one is reducible to the maximal one. Notice that if there is any unifier at all, we must have

$$A \equiv r(a_1, \ldots, a_k), \quad B \equiv r(b_1, \ldots, b_k), \tag{85}$$

i.e., A and B must be terms involving the same pf variable.

At the basis, none of the variables occur, so there is only one possible unifier, the empty τ, σ, which reassigns no variables; it is a unifier exactly when the two expressions are identical and it is trivially maximal. We take this case of expressions of the form $\lambda(\,)A$, $\lambda(\,)B$ to cover also (by convention) the case of terms A, B.

For the induction step, suppose u_i actually occurs in A. We distinguish cases.

CASE 1. For some j, $a_j \equiv u_i$ but b_j is none of the v_t's.

In this case every unifier must set $\tau(u_i) \equiv b_j$, we make this replacement on the expressions and we get the result by applying the induction hypothesis to these new expressions which have smaller critical number since u_i does not occur in them.

CASE 2. For some j, $a_j \equiv u_i$, and for every j, if $a_j \equiv u_i$ then b_j is one of the v_t's, but there exist distinct $j \neq j'$ such that $a_j \equiv a_{j'} \equiv u_i$ and $b_j \not\equiv b_{j'}$.

In this case every unifier must satisfy

$$\tau(u_i) \equiv \sigma(b_j) \equiv \sigma(b_{j'}),$$

and we can apply the ind. hyp. to the expressions resulting by identifying b_j with $b_{j'}$ on the right, which have smaller critical number.

CASE 3. u_i occurs in A, and for all j and some t,

$$a_j \equiv u_i \Longrightarrow b_j \equiv v_t.$$

In this case we look at the symmetric argument, taking Cases 1, 2 and 3 on v_t. In the first of these two we can use the induction hypothesis and the last leads us to the following:

CASE 4. u_i occurs in A and there exists some v_t which occurs in B at precisely all the same places where u_i occurs in A.

In this case every unifier must set $\tau(u_i) \equiv \sigma(v_t)$, and this variable may be one of the bound or the free ones in the final expressions. We choose a variable s not occurring anywhere, we replace u_i in A and v_t in B by s throughout and we apply the ind. hyp. to these new expressions with lower critical number to obtain some maximal bvu (if it exists) $(\tau, \sigma, (s_1, \ldots, s_k))$; the maximal bvu for the

original expressions is obtained by adding s to the bound variables and extending τ and σ in the obvious way,

$$\tau(u_i) \equiv \sigma(v_t) \equiv s. \qquad \dashv$$

Suppose now we are given an identity (63). We first show how to construct a single dictionary line from it, which will determine the strict truth of (63) when all the free, basic variables in it are interpreted by distinct values in the domain. The complete set of lines for (63) will contain the lines we get in this way from all the identities which result from (63) by the identification of some of its free, basic variables.

We assume that the special variables $x_1, \ldots, x_m$ we will use for the lines do not occur in the given identity.

STEP 1. For each z_i which is a basic variable, we put in the antecedent of the line the equality $x_i = z_i$ and the condition $x_i\downarrow$.

STEP 2. Consider any pair z_i, z_j $(i \neq j)$ of expressions which are not basic variables; we will view these as λ-expressions by identifying temporarily a term $r(\vec{x})$ with the λ-expression $\lambda(\,)r(\vec{x})$. If z_i, z_j cannot be unified, we add nothing to the line. In the opposite case, suppose

$$z_i \equiv \lambda(u_1, \ldots, u_{arity(i)})r(x_1, \ldots, x_k), \quad z_j \equiv \lambda(v_1, \ldots, v_{arity(j)})r(y_1, \ldots, y_k),$$

and $(\tau, \sigma, (s_1, \ldots, s_l))$ is a maximal bvu for these expressions. We add to the antecedent of the line the equation

$$\lambda(s_1, \ldots, s_l)x_i(\tau(u_1), \ldots, \tau(u_{arity(i)})) \simeq \lambda(s_1, \ldots, s_l)x_j(\sigma(v_1), \ldots, \sigma(v_{arity(j)})). \tag{86}$$

STEP 3. After Steps 1 and 2 are completed, we add to the antecedent of the line the inequality $u \neq v$ and the conditions $u\downarrow$, $v\downarrow$, for every two free, basic variables which occur free on the line and are not among the standard $x_1, \ldots, x_n$. (If only one such u occurs, we simply add $u\downarrow$.)

To recapitulate what we said above, the complete set of lines associated with an identity is obtained by applying this procedure to every identity obtained by identifying some or all of the free basic variables of the identity.

To illustrate the procedure, consider again (77). There are no terms to unify here, only Steps 1 and 3 come up and we get the following two lines:

$$x_1 = x,\ x_2 = y,\ x_3 = x,\ x_1\downarrow,\ x_2\downarrow,\ x_3\downarrow,\ x \neq y,\ x\downarrow,\ y\downarrow \implies f(x_1, x_2) = g(x_3),$$
$$x_1 = x,\ x_2 = x,\ x_3 = x,\ x_1\downarrow,\ x_2\downarrow,\ x_3\downarrow,\ x\downarrow \implies f(x_1, x_2) = g(x_3).$$

The procedure generates only one line for (78):

$$x_1 = x,\ x_3 = x,\ x_1\downarrow,\ x_3\downarrow \implies f(x_1, x_2) = g(x_3).$$

Consider also the example

$$f(\lambda(ts)p(x, y, t, s), q(x), x) \simeq g(p(x, y, x, y), q(y), x). \tag{87}$$

There are two free variables, so we will get two lines. First from the identity as it is, Step 2 will come into play with the most general bvu for z_1 and z_3 obviously being

$$\tau : t := x,\ s := y,$$

with no bound variables left, so the line produced is

$$x_3 = x,\ x_3{\downarrow},\ x_6 = x,\ x_6{\downarrow},\ x_1(x,\ y) = x_4,\ x \neq y,\ x{\downarrow},\ y{\downarrow} \Longrightarrow f(x_1, x_2, x_3) = g(x_4,\ x_5, x_6).$$

If we identify $x \equiv y$, we get the identity

$$f(\lambda(ts)p(x, x, t, s), q(x), x) \simeq g(p(x, x, x, x), q(x), x)$$

which has an additional (trivial) unification and generates the line

$$x_3 = x,\ x_3{\downarrow},\ x_6 = x,\ x_6{\downarrow},\ x_1(x, x) = x_4,\ x_2 = x_5,\ x{\downarrow} \Longrightarrow f(x_1, x_2, x_3) = g(x_4,\ x_5, x_6).$$

It is quite easy to verify directly that the extended validity of these two lines is equivalent to the strict validity of the identity.

4.13. LEMMA. *The conjunction of all the lines constructed for* (63) *implies* (63).

PROOF. We will verify that if the line produced by an identity holds, then every strict assignment which assigns distinct values to distinct basic variables satisfies the identity, from which the Lemma follows by applying it to all the substitution instances of the identity.

Suppose we are given a strict assignment π to the variables of the identity and extend it to the special variables which occur on the line by setting

$$\pi(x_i) = \pi(z_i). \tag{88}$$

It will be enough to verify that this assignment satisfies the antecedent of the line, so consider how the clauses were introduced to it by the three steps of the construction.

STEP 1. We put in $x_i = z_i$ and $x_i{\downarrow}$ if z_i is basic, and π clearly makes these conditions true, by definition and because it is strict on the variables which occur free in the identity.

STEP 2. If the clause (86) is added to the antecedent of the line in this step, we must show that it is validated by π, or equivalently that π validates the term identity

$$x_i(\tau(u_1), \ldots, \tau(u_{arity(i)})) \simeq x_j(\sigma(v_1), \ldots, \sigma(v_{arity(j)})). \tag{89}$$

We compute:

$$\begin{aligned}
\pi(x_i(\tau(u_1), \ldots, \tau(u_{arity(i)}))) &\simeq \pi(z_i)(\tau(u_1), \ldots, \tau(u_{arity(i)})) \\
&\simeq \pi((\lambda(u_1, \ldots, u_m)r(a_1, \ldots, a_k))(\tau(u_1), \ldots, \tau(u_{arity(i)}))) \\
&\simeq \pi(r(\tau(a_1), \ldots, \tau(a_k))) \\
&\simeq \pi(r(\sigma(b_1), \ldots, \sigma(b_k))),
\end{aligned}$$

where the last line follows from the fact that τ, σ are parts of a unifier. From this point we proceed with the opposite computation for z_j, to complete the proof of (89).

STEP 3. The clauses introduced by Step 3 are obviously validated by π, which is assumed to assign distinct, strict values to distinct basic variables. ⊣

4.14. LEMMA. *An identity implies the validity of every line it generates.*

PROOF. We now assume that we are given an extended assignment π to the variables of the line which satisfies the antecedent, and we must show that it also satisfies the consequent.

Notice that π assigns distinct values other than $\perp$ to all the basic variables which occur free both in the line and in the identity, because of the clauses we introduced in Step 3. We extend π to the remaining free basic variables in the equation by giving a distinct, new value to each of them. We want to extend π to all the pf variables also, so that we get

$$\pi(z_i) \simeq \pi(x_i), \tag{90}$$

for every $i = 1, \ldots, m$. This is already true when z_i is a basic variable, because of the equations put in the line in Step 1.

To effect (90), for

$$z_i \equiv \lambda(u_1, \ldots, u_{arity(i)})r(a_1, \ldots, a_k),$$

we want to define $\pi(r)$ so that

$$\pi(r)(\pi^*(a_1), \ldots, \pi^*(a_k)) \simeq \pi(x_i)(\pi^*(u_1), \ldots, \pi^*(u_{arity(i)})) \tag{91}$$

for every π^* which agrees with π on all the variables except (perhaps) $u_1, \ldots, u_{arity(i)}$. This will be possible, unless there is another expression

$$z_j \equiv \lambda(v_1, \ldots, v_{arity(j)})r(b_1, \ldots, b_k)$$

which then demanded

$$\pi(r)(\pi^*(b_1), \ldots, \pi^*(b_k)) \simeq \pi(x_j)(\pi^*(v_1), \ldots, \pi^*(v_{arity(j)})), \tag{92}$$

but for some assignment π^* as above we have the conflict

$$\pi^*(a_1), \ldots, \pi^*(a_k) \simeq \pi^*(b_1), \ldots, \pi^*(b_k), \tag{93}$$

$$\pi(x_i)(\pi^*(u_1), \ldots, \pi^*(u_{arity(i)})) \neq \pi(x_j)(\pi^*(v_1), \ldots, \pi^*(v_{arity(j)})). \tag{94}$$

First we argue that if (93) holds, then the terms z_i and z_j can be unified. Let $w(1), \ldots, w(l)$ be the distinct values that π^* assigns to the variables $u_1, \ldots, u_{arity(i)}$, $v_1, \ldots, v_{arity(j)}$ (it maybe that $l = 0$), choose distinct, fresh variables $s(1), \ldots, s(l)$ and set

$$\tau(u_\alpha) \equiv \begin{cases} a_\beta, \text{ if } \pi^*(u_\alpha) = \pi(a_\beta), \\ s(w(\pi^*(u_\alpha))), \text{ otherwise,} \end{cases}$$

where a_β is meant to be a variable not among $u_1, \ldots, u_{arity(i)}$. Notice that if the first case of the definition applies, then the variable a_β is uniquely determined, because π^* agrees with π on these variables and assigns a distinct value to each of them. We give a similar definition of another variable transformation σ on the v_α's and then observe that because of (93), these transformations define a unifier of z_i with z_j with the bound variables $s_1, \ldots, s_l$.

Using Lemma **4.12**, we can find a maximal unifier of z_i, z_j from which the one just constructed can be obtained by further specifying or identifying some of the bound variables. It follows that in Step 2 of the construction of the line we put in a clause which expresses the maximal unification of z_i with z_j and this easily contradicts (94). ⊣

4.15. Structures with (some) finite basic sets. There is an obvious, trivial way by which we can reduce the problem of intensional identity for any structure $\mathbf{A}$ of finite signature to that of another such structure $\mathbf{A}'$ in which all basic sets are infinite. If, for example, $U = \{u_1, \ldots, u_n\}$ is finite and $f : U \times V \rightharpoonup W$ is a partial function among the givens, we replace f in $\mathbf{A}$ by n new partial functions

$$f_i : V \rightharpoonup W, \quad f_i(v) \simeq f(u_i, v), \quad i = 1, \ldots, n.$$

The translation is a bit messier for functionals but still trivial in principle. This is, in fact what we will do if U is a *small* finite set, e.g., if $U = TV = \{\mathbf{t}, \mathbf{f}\}$ is the set of truth values. If, however, n is immense, then this reduction leads to a structure with impossibly many primitives whose dictionary is totally unmanageable. Suppose, for example, that the language is a small (in theory formalized) fragment of the basic English currently in use as the common language of business in continental Europe. There are few basic sets in the intended interpretation, the citizens of France, the German cars, the Greek raisins, etc., but they are all immense. We may also assume few primitives, we are only interested in making simple assertions like

there are enough Greek raisins to satisfy the needs of all Frenchmen.

The problem of sense identity for sentences of such a language appears to be quite manageable, and in fact, the actual dictionaries we would use to translate this language into the national European languages are quite small. In contrast, the formal dictionary of the expanded language suggested by the trivial procedure of eliminating all the finite basic sets is absurdly large and involves specific entries detailing separately the relation between every Frenchman with every Greek raisin. The decision procedure we described allows a better solution.

4.16. COROLLARY (to the proof). *Suppose* $\mathbf{A} = (U_1, \ldots, U_k, f_1, \ldots, f_n)$ *is a recursor structure of finite signature, such that every basic set* U_i *has at least* d *members. Then the decision procedure for intensional identity defined in this section will decide correctly every identity on* $\mathbf{A}$ *with* n *(free and bound) basic variables, provided that* $2n + 4 \leq d$.

The Corollary suggests a method of constructing a reasonably sized "dictionary of meanings" for a structure in which some basic sets are very small—and we eliminate these—and the others are very large. The formal decision procedure for intensional identity on the basis of this dictionary is not that far from the way we would decide such questions in practice: we understand quantification over small sets by considering individual cases, while for large sets we appeal to fundamental identities relating the meanings of the primitives, including the quantifiers. The procedure will fail to resolve questions of identity of meaning which involve more quantifiers over large sets than (roughly) half the size of the structure. The proof of the Corollary follows from a careful examination of the arguments of this section, which basically require the existence of enough possible values for variables to make certain distinctions. For example, it is not hard to check that Lemma **4.3** holds, provided all the basic sets of the structure have at least 4 elements. We will omit the details.

References

[1] A. Church, *Intensionality and the paradox of the name relation.* To appear in the Proceedings of the meeting at SUNY, Buffalo.

[2] ——, *A formulation of the logic of sense and denotation, abstract*, ***Journal of Symbolic Logic***, **11** (1946), p. 31.

[3] ——, *A formulation of the logic of sense and denotation*, in ***Structure, Method and Meaning***, P. Henle, H. M. Kallen, and S. K. Langer, eds., Liberal Arts Press, New York, 1951, pp. 3–24.

[4] ——, *Outline of a revised formulation of the logic of sense and denotation, part I*, ***Noûs***, **7** (1973), pp. 24–33.

[5] ——, *Outline of a revised formulation of the logic of sense and denotation, part II*, ***Noûs***, **8** (1974), pp. 135–156.

[6] M. J. Cresswell, ***Structured Meanings: The Semantics of Propositional Attitudes***, The MIT Press, Cambridge, Mass, 1985.

[7] K. Donnellan, *Reference and definite description*, ***Philosophical Review***, **75** (1966), pp. 281–304.

[8] M. A. Dummett, *Frege's distinction between sense and reference*, in ***Truth and Other Enigmas***, Harvard Univ. Press, Cambridge, 1978, pp. 116–144.

[9] G. Evans, ***The Varieties of Reference***, Clarendon Press, Oxford, 1982. Edited by J. N. McDowell.

[10] J. E. Fenstad, R. O. Gandy, and G. E. Sacks, eds., ***Generalized Recursion Theory, II***, Studies in Logic, No. 94, North Holland, Amsterdam, 1978.

[11] J. E. Fenstad and P. G. Hinman, eds., ***Generalized Recursion Theory***, Studies in Logic, No. 79, North Holland/American Elsevier, Amsterdam, 1974.

[12] G. Frege, *On sense and denotation*, in ***Translations from the Philosophical Writings of Gottlob Frege***, P. Geach and M. Black, eds., Basil Blackwell, Oxford, 1952. Translated by Max Black under the title *Sense and meaning.* In quotations from Black's translation, I have consistently changed the rendering of *Bedeutung* as *meaning* to the more current *denotation* or *reference.*

[13] A. S. Kechris and Y. N. Moschovakis, *Recursion in higher types*, in ***Handbook of Mathematical Logic***, J. Barwise, ed., Studies in Logic, No. 90, North Holland, Amsterdam, 1977, pp. 681–737.

[14] S. C. KLEENE, *Recursive functionals of finite type, I*, ***Transactions of the American Mathematical Society***, **91** (1959), pp. 1–52.

[15] ——, *Recursive functionals of finite type, II*, ***Transactions of the American Mathematical Society***, **108** (1963), pp. 106–142.

[16] D. E. KNUTH, ***Fundamental Algorithms***, vol. 1 of The Art of Computer Programming, Addison-Wesley, 1968.

[17] G. KREISEL and G. TAKEUTI, *Formally self-referential propositions for cut-free classical analysis and related systems*, ***Dissertationes Mathematicae***, **118** (1974).

[18] S. A. KRIPKE, ***Naming and Necessity***, Harvard Univ. Press, Cambridge, 1972.

[19] ——, *Outline of a theory of truth*, ***Journal of Philosophy***, **72** (1975), pp. 690–716.

[20] R. MONTAGUE, *Universal grammar*, ***Theoria***, **36** (1970), pp. 373–398.

[21] Y. N. MOSCHOVAKIS, ***Elementary Induction on Abstract Structures***, Studies in Logic, No. 77, North Holland, Amsterdam, 1974.

[22] ——, *Abstract recursion as a foundation of the theory of recursive algorithms*, in ***Computation and Proof Theory***, M. M. Richter et al., eds., Lecture Notes in Mathematics, No. 1104, Springer-Verlag, Berlin, 1984, pp. 289–364.

[23] ——, *The formal language of recursion*, ***Journal of Symbolic Logic***, **54** (1989), pp. 1216–1252.

[24] ——, *A mathematical modeling of pure recursive algorithms*, in ***Logic at Botik '89***, A. R. Meyer and M. A. Taitslin, eds., Lecture Notes in Computer Science, No. 363, Springer-Verlag, Berlin, 1989, pp. 208–229.

[25] R. PLATEK, ***Foundations of Recursion Theory***, Ph.D. thesis, Stanford University, 1966.

[26] B. RUSSELL, *On denoting*, ***Mind***, **14** (1905), pp. 479–493.

[27] G. E. SACKS, ***Higher Recursion Theory***, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1990.

[28] A. M. TURING, *On computable numbers, with an application to the Entscheidungsproblem*, ***Procedings of the London Mathematical Society,*** Series II, **42** (1936–37), pp. 230–265.

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

[30] ——, *Sense in Frege*, in ***Selected Essays*** [29], pp. 55–64.

Department of Mathematics
University of California, Los Angeles
ynm@math.ucla.edu