11 Linguistic and Computational Semantics

Abstract

I argue that because the very concept of computation rests on notions of interpretation, the semantics of natural languages and the semantics of computational formalisms are in the deepest sense the same subject. The attempt to use computational formalisms in aid of an explanation of natural language semantics, therefore, is an enterprise that must be undertaken with particular care. I describe a framework for semantical analysis that I have used in the computational realm, and suggest that it may serve to underwrite computationally-oriented linguistic semantics as well. The major feature of this framework is the explicit recognition of both the declarative and the procedural import of meaningful expressions; I argue that whereas these two viewpoints have traditionally been taken as alternative, any comprehensive semantical theory must account for how both aspects of an expression contribute to its overall significance.

A1

I have argued elsewhere[1] that the distinguishing mark of those objects and processes we call computational has to do with **attributed semantics**: we humans find computational processes coherent exactly because we attach semantical significance

A2

1. Smith (1982b).

to their behavior. ingredients, and so forth. Put another way, computers, on this view, are those devices that we understand by *deploying our linguistic facilities*. For example. the reason that a calculator is a computer, but a car is not, is that we take the ingredients of the calculator to be *symbolic* (standing, in this particular case, for numbers and functions and so forth), and understand the interactions and organisation of the calculator in terms of that interpretation (this part *divides*, this part *represents the sum*, and so on). Even though by and large we are able to produce an explanation of the behavior that does not rest on external semantic attribution (this is the *formality condition* mentioned by Fodor, Haugeland, and others[2]), we nonetheless speak, when we use computational terms, in terms of this semantics. These semantical concepts rest at the foundations of the discipline: the particular organisations that computers have—their computational *raison d'être*—emerge not only from their mechanical structure but also from their semantic interpretability. Similarly, the terms of art employed in computer science—*program*, *compiler*. *implementation*, *interpreter*, and so forth—will ultimately be definable only with reference to this attributed semantics; they will not, in my view, ever be found reducible to non-semantical predicates.[3]

A3 A4 A5 A6

This is a ramifying and problematic position, which I cannot defend here.[4] I may simply note, however, the overwhelming evidence in favour of a semantical approach manifested

2. Fodor (1978), Fodor (1980), Haugeland (forthcoming).

3. At least until the day arrives—if ever—when a successful psychology of language is presented wherein *all* of human semanticity is explained in non-semantical terms.

4. Problematic because it defines computation in a manner that is derivative on mind (in that language is fundamentally a mental phenomenon), thus dashing the hope that computational psychology will offer a release from the semantic irreducibility of previous accounts of human cognition. Although I state this position and explore some of its consequences in Smith (1982b), a considerably fuller treatment will be provided in Smith (forthcoming).

A6

by everyday computational language. Even the simple view of computer science as the study of *symbol manipulation*[5] reveals this bias. Equally telling is the fact that programming languages are called *languages*. In addition, language-derived concepts like *name* and *reference* and *semantics* permeate computational jargon (to say nothing of *interpreter*, *value*, *variable*, *memory*, *expression*, *identifier* and so on)—a fact that would be hard to explain if semantics were not crucially involved. It is not just that in discussing computation we use language; rather, in discussing computation we use words that suggest that we are also talking *about* linguistic phenomena.

The question I will focus on in this paper, very briefly, is this: if computational artefacts are fundamentally linguistic, A8
and if, therefore. it is appropriate to analyze them in terms of formal theories of semantics (it is apparent that this is a wide- A9
ly held view), then what is the proper relationship between the so-called *computational* semantics that results, and more standard *linguistic* semantics (the discipline that studies people and their natural languages: how we mean, and what we are talking about, and all of that good stuff)? And furthermore, what is it to *use* computational models to *explain* natural language semantics, if the computational models are themselves in need of semantical analysis? On the face of it, there would seem to be a certain complexity that should be sorted out. A10

In answering these questions I will argue approximately as follows: in the limit computational semantics and linguistic semantics will coincide, at least in underlying conception, if not in surface detail (for example some issues, like ambiguity, may arise in one case and not in the other). Unfortunately, however, as presently used in computer science the term 'semantics' is given such an operational cast that it distracts attention from the human attribution of significance to computational structures.[6] In contrast, the most successful models of natural A11

5. See for example Newell (1980).
6. The term "semantics" is only one of a large collection of terms, un-

language semantics, embodied for example in standard model theories and even in Montague's program, have concentrated almost exclusively on *referential* or *denotational* aspects of declarative sentences. Judging only by surface use, in other words, computational semantics and linguistic semantics appear almost orthogonal in *concern*, even though they are of course similar in *style* (for example they both use meta-theoretic mathematical techniques—functional composition, and so forth—to recursively specify the semantics of complex expressions from a given set of primitive atoms and formation rules). It is striking, however, to observe two facts. First, computational semantics is being pushed (by people and by need) more and more towards declarative or referential issues. Second, natural language semantics, particularly in computationally-based studies, is focusing more and more on pragmatic questions of use and psychological import. Since computational linguistics operates under the computational hypothesis of mind, psychological issues are assumed to be modelled by a field of computational structures and the state of a processor running over them; thus these linguistic concerns with "use" connect naturally with the "operational" flavour of standard programming language semantics. It seems not implausible, therefore—I betray caution with the double negative—that a unifying framework might be developed.

A12 A13 A14

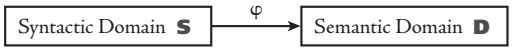

Figure 1 — Traditional (simple) semantic model

fortunately, that are technical terms in computer science and in the attendant cognitive disciplines (including logic, philosophy of language, linguistics, and psychology), with different meanings and different connotations. *Reference*, *interpretation*, *memory*, and *value* are just a few examples of the others. It is my view that in spite of the fact that semantical vocabulary is used in *different* ways, the fields are both semantical in fundamentally the *same* ways: a unification of terminology would only be for the best.

A15

It will be the intent of this paper to present a specific, if preliminary, proposal for such a framework. First, however, some introductory comments. In a general sense of the term, *semantics* may be taken as the study of the relationship between entities or phenomena in a syntactic domain S and corresponding entities in a semantic domain D, as pictured in figure 1.

I will call the function mapping elements from the first domain into elements of the second an **interpretation function** (to be sharply distinguished[7] from what in computer science is called an *interpreter*, which is a different beast altogether). A16 Note that the question of whether an element is syntactic or semantic is a function of the point of view; the syntactic domain for one interpretation function can readily be the semantic domain of another (and a semantic domain may of course include its own syntactic domain).

Not all relationships, of course, count as semantical; the "grandmother" relationship fits into the picture just sketched, but stakes no claim on being semantical. Though it has often been discussed what constraints on such a relationship characterize genuinely semantical ones (compositionality or recursive specifiability, and a certain kind of formal character to the syntactic domain, are among those typically mentioned), I will not pursue such questions here. Rather, I will complicate the diagram as indicated in figure 2, so as to enable us to characterize a rather large class of computational and linguistic formalisms. A17

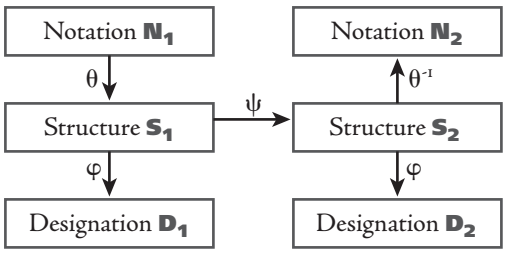

Figure 2 — More general semantic model

N1 and N2 are intended to be *notational* or *communicational expressions*, in some externally observable and consensually

7. An example of the phenomenon noted in note 6.

established medium of interaction, such as strings of characters, streams of words, or sequences of display images on a computer screen. The relationship θ is an interpretation function mapping notations into *internal elements* of some process over which the primary semantical and processing regimens are defined. In first-order logic, s_1 and s_2 would be something like abstract derivation tree types of first-order formulae; if the diagram were applied to the human mind, under the hypothesis of a formally encoded mentalese, s_1 and s_2 would be tokens of internal mentalese, and θ would be the function computed by the "linguistic" faculty (on a view such as that of Fodor[8]). In adopting these terms I mean to be speaking *very* generally; thus I mean to avoid, for example, any claim that *tokens* of English are **internalized** (a term I will use for θ) into recognizable tokens of mentalese. In particular. the proper account of θ for humans could well simply describe how the field of mentalese structures, in some configuration, is transformed into some other configuration, upon being presented with a particular English sentence; this would still count, on this view, as a theory of θ. A18

In contrast, φ is the interpretation function that makes explicit the standard denotational significance of linguistic terms, relating, we may presume, expressions in $\mathcal{S}$ to the world of discourse. The relationship between my mental token for T. S. Eliot, for example, and the poet himself, would be formulated as part of φ. Again, I am speaking very broadly; φ is intended to manifest what, paradigmatically, expressions are *about*, however that might best be formulated (φ includes for example the interpretation functions of standard model theories). ψ, in contrast, relates some internal structures or states to others—one can imagine it specifically as the formally computed derivability relationship in a logic (⊢), as the function computed by the primitive language processor in a computational machine (i.e., as Lisp's EVAL), or more generally as the

8. Fodor (forthcoming)

function that relates one configuration of a field of symbols to another, in terms of the modifications engendered by some internal processor computing over those states. (φ and ψ are named, for mnemonic convenience, by analogy with *philosophy* and *psychology*, since a study of φ is a study of the relationship between expressions and the world—since philosophy takes you "out of your mind," so to speak—whereas a study of ψ is a study of the internal relationships between symbols, all of which, in contrast, are "within the head" of the person or machine.)

Some simple comments. First, N_1, N_2, S_1, S_2, D_1, and D_2 need not all necessarily be distinct: in a case where S_1 is a self-referential designator, for example, D_1 would be the same as S_1; similarly, in a case where ψ computed a function that was designation-preserving, then D_1 and D_2 would be identical. Secondly, we need not take a stand on which of ψ and φ has a prior claim to being the *semantics* of S_1. In standard logic, ψ (i.e., derivability: ⊢) is a relationship, but is far from a function, and there is little tendency to think of it as *semantical*; a study of ψ is called *proof theory*. In computational systems, on the other hand, ψ is typically much more constrained, and is also, by and large, analyzed mathematically in terms of functions and so forth, in a manner much more like standard model theories. Although in my own view it seems a little far-fetched to call the internal relationships (the "use" of a symbol) *semantical*, it is nonetheless true that we are interested in characterizing both, and it is unnecessary to express an preference. For discussion, I will refer to the φ-semantics of a symbol or expression as its **declarative import**, and refer to its ψ-semantics as its **procedural consequence**. I have heard it said in other quarters that "procedural" and "declarative" theories of semantics are contenders;[9] to the extent that I have been able to make sense of these notions, it appears that we need both.

It is possible to use figure 2 to characterize a variety of stan-

9. Woods (1981)

dard formal systems. In the standard models of the λ-calculus. for example, the designation function φ takes λ-expressions onto functions; the procedural regimen ψ, usually consisting of α- and β-reductions, can be shown to be φ-preserving. Similarly, if in a standard predicate logic we take φ to be (the inverse of the) satisfaction relationship, with each element of $\mathcal{S}$ being a sentence or set of sentences, and elements of $\mathcal{D}$ being those possible worlds in which those sentences are true, and similarly take ψ as the derivability relationship, then soundness and completeness can be expressed as the equation $\psi(s_1, s_2) \equiv [D_1 \subseteq D_2]$. As for all *formal* systems (these presumably subsume the computational ones), it is crucial that ψ be specifiable independent of φ. The λ-calculus and predicate A19
logic systems, furthermore, have no notion of a processor with state; thus the appropriate ψ involves what we may call **local procedural consequence**, relating a simple symbol or set of symbols to another set. In a more complex computational circumstance, as I will show below, it is appropriate to characterize a more complex **full procedural consequence** involving not only simple expressions, but fuller encodings of the state of various aspects of the computational machine (for example. at least environments and continuations in the typical computational case[10]).

An important consequence of the analysis illustrated in figure 2 is that it enables one to ask a question not typically asked in computer science, about the (φ-) *semantic character* of the function computed by ψ. Note that questions about soundness and completeness in logic are exactly questions of this type. In separate research,[11] I have shown, by subjecting [them] to this kind of analysis, that computational formalisms can be usefully analyzed in these terms as well. In particular, A20

10. For a discussion of continuations see Gordon (1979), Steele and Sussman (1978), and Smith (1982a) [see ch. 3]; the formal device is developed in Strachey & Wadsworth (1974).
11. Smith (1982a) [see ch. 3].

I demonstrated that the universally accepted Lisp evaluation protocol is semantically confused, in the following sense: sometimes it preserves φ (i.e. φ(ψ(s))=φ(s)), and sometimes it *embodies* φ (thereby "de-referencing" its inputs: ψ(s)=φ(s). The traditional Lisp notion of evaluation. in other words, conflates *simplification* and *reference* relationships, to its peril (in that report I propose some Lisp dialects in which these two notions are kept much more neatly and strictly separate). The current moral, however. is merely that this approach allows the question of the semantical import of ψ to be asked.

As well as considering Lisp, we may use our diagram to characterize various linguistically oriented projects carried on under the banner of "semantics." Model theories and formal theories of language (I am including Tarski and Montague in one sweep) have concentrated primarily on φ. Natural language semantics in some quarters[12] focuses on θ—on the "translation" of natural language into an internal medium—although the question of what aspects of a given sentence must be preserved in such a translation are of course of concern (no translator could ignore the salient properties, semantical and otherwise, of the target language, be it mentalese or predicate logic, since the endeavour would otherwise be without constraint). Lewis (for one) has argued that the project of articulating θ—an endeavour he calls *markerese* semantics—cannot really be called semantics at all,[13] since it is essentially a translation relationship, although it is worth noting that θ in computational formalisms is not always trivial, and a case can at least be made that many superficial aspects of natural language use, such as the resolution of indexicals, may be resolved at this stage (if for example you say "I am warm" then I may internalise your use of the first person pronoun into my internal name for you). A21

12. A classic example is Katz and Postal (1964), but much of the recent AI research in natural language can be viewed in this light.
13. Lewis (1972).

Those artificial intelligence researchers working in knowledge representation, perhaps without too much distortion, can be divided into two groups: (i) those whose primary semantical allegiance is to φ, and who (perhaps as a consequence) typically use an encoding of first-order logic as their representation language; and (ii) those who concern themselves primarily with ψ, and who therefore (legitimately enough) reject logic as even suggestive (ψ in logic—derivability—is a relatively unconstrained relationship, for one thing; secondly, the relationship between the entailment relationship ⊨, to which derivability is a hopeful approximation, and the proper "ψ" of rational belief revision, is at least a matter of debate[14]).

Programming language semantics, for reasons that can at least be explored, if not wholly explained, have focused primarily on ψ, although in ways that tend to confuse it with φ. Except for Prolog, which borrows its φ straight from a subset of first-order logic, and the [reconstructed] Lisps mentioned earlier,[15] I have never seen a semantical account of a programming language that gave *independent* accounts of φ and ψ. A22 There are complexities, furthermore, in knowing just what the proper treatment of general languages should be. In a separate paper[16] I argue that the notion *program* is inherently defined as a set of expressions whose (φ-) semantic domain includes *data structures* (and set-theoretic entities built up over them). In other words, in a computational process that deals with finance, say, the general data structures will likely designate individuals and money and relationships among them, but the terms in that part of the process called a *program* will not designate these people and their money, but will instead designate *the data structures that designate people and money* (plus of course relationships and functions over those data structures). A23 Even on a *declarative* view like mine, in other words, the ap-

14. Israel (1980).
15. For a discussion of Prolog see Clocksin & Mellish (1981); the Lisps are described in Smith (1982a) [see ch. 3].
16. [See ch. 3].

propriate semantic domain for programs is built up over data structures—a situation strikingly like the standard semantical accounts that take abstract records or locations or whatever as elements of the otherwise mathematical domain for programming language semantics. It may be that this fact that all base terms in programs are *meta-syntactic* that has spawned the confusion between operations and reference in the computational setting. A24

Although the details of a general story remain to be worked out, the Lisp case mentioned earlier is instructive, by way of suggestion as to how a more complete computational theory of language semantics might go. In particular, because of the context relativity and non-local effects that can emerge from processing a Lisp expression, φ is not specifiable in a strict compositional way. ψ—when taken to include the broadest possible notion that maps entire configurations of the field of symbols and of the processor itself onto other configurations and states—is of course recursively specifiable (the same fact, in essence, as saying that Lisp is a deterministic formal calculus). A pure characterization of ψ *without* a concomitant account of φ, however, is unmotivated—as empty as a specification of a derivability relationship would be for a calculus for which no semantics had been given. Of more interest is the ability to specify what I call a **general significance function** Σ, which recursively specifies ψ and φ together (this is what I was able to do for Lisp). In particular, given any expression s_1, any configuration of the rest of the symbols, and any state of the processor, the function Σ will specify the configuration and state that would result (i.e., it will specify the *use* of s_1), and also the relationship to the world that the whole signifies. For example, given a Lisp expression of the form (+ 1 (PROG (SETQ A 2) A)), Σ would specify that the whole expression designated the number three, that it would return the numeral '3', and that the machine would be left in a state

in which the binding of the variable A was changed to the numeral '2'. A modest result; what is important is merely (i) that both declarative import and procedural significance must be reconstructed in order to tell the full story about Lisp; and (ii) that they must be formulated together.

Rather than pursue this view in detail, it is helpful to set out several points that emerge from analyses developed within this framework:

1. In most programming languages, θ can be specified compositionally and independently of φ or ψ—this amounts to a formal statement of Fodor's *modularity thesis* for language.[17] In the case of *formal* systems, θ is often context-free and compositional, but not always (*reader macros* can render it opaque, or at least intensional, and some languages such as Algol are apparently context-sensitive). It is noteworthy. however. that there have been computational languages for which θ could not be specified independently of ψ—a fact that is often stated as the fact that the programming language "cannot be parsed except at runtime" (Teco and the first versions of Smalltalk had this character). A25
2. Since Lisp is computational, it follows that a *full* account of its ψ can be specified independent of ϕ; this is in essence the formality condition. It is important to bring out, however. that a *local* version of ψ will typically not be compositional in a modem computational formalism, even though such locality holds in purely extensional context-free side-effect free languages such as the λ-calculus. A26 A27
3. It is widely agreed that ψ does not uniquely determine φ (this is the "psychology narrowly construed" and the concomitant methodological solipsism of Putnam and Fodor and others[18]). However this fact is compatible

17. Fodor (forthcoming).
18. The term "methodological solipsism" is from Putnam (1975); see also

with our foundational claim that computational systems are distinguished in virtue of having some version of φ as part of their characterization. A very similar point can be made for logic: although any given logic can (presumably) be given a mathematically-specified model theory, that theory does not typically tie down what is often called the *standard* model or interpretation—the interpretation that we use. This fact does not release us, however, from positing as a candidate logic only a formalism that humans can interpret. A28

4. The declarative interpretation [function] φ cannot be wholly determined independent of ψ, except in purely declarative languages (such as the λ-calculus and logic and so forth). This is to say that without some account of the effect on the processor of one fragment of a whole linguistic structure, it may be impossible to say what that processor will take another fragment as designating. The use of SETQ in Lisp is an example; natural language instances will be explored below.

This last point needs a word of explanation. It is of course possible to specify φ in mathematical terms without any explicit mention of a ψ-like function; the approach I use in Lisp defines both ψ and φ in terms of the overarching function Σ mentioned above, and I could of course simply define φ without defining ψ at all. My point, rather, is that any successful definition of φ will effectively have to do the work of ψ, more or less explicitly, either by defining some identifiable relationship, or else by embedding that relationship within the meta-theoretic machinery. I am arguing, in other words, only that the *subject* I intend ψ to cover must be treated in some fashion or other.

What is perhaps surprising about all of this machinery is that it must be brought to bear on a purely procedural language—all three relationships (θ, φ, and ψ) figure crucially

. . .

Fodor (1980).

in an account even of Lisp. I are not suggesting that Lisp is *like* natural languages; to point out just one crucial difference, there is no way in Lisp or in any other programming language (except Prolog) to *say* anything, whereas the ability A29
to say things is clearly a foundational aspect of any human language. The problem in the procedural languages is one of what we may call **assertional force:** although it is possible to construct a sentence-like expression with a clear declarative semantics (such as some equivalent of "x=3"), one cannot use it in such a way as to actually mean it—so as to have it carry any assertional weight. That is, it is trivial to set some variable x to 3, or to ask whether x is 3, but there is no way to state *that x is 3*. It should be admitted, however, that computational languages bearing assertional force are under considerable current investigation. This general interest is probably one of the reasons for Prolog's emergent popularity; other computational systems with an explicit declarative character include for example specification languages, data base models, constraint languages, and knowledge representation languages in Artificial Intelligence (AI). We can only assume that the appropriate semantics for all of these formalisms will align even more closely with an illuminating semantics for natural language.

What does all of this have to do with natural language, and with computational linguistics? The essential point is this: *if* this characterization of formal systems is tenable, and if the techniques of standard programming language semantics can be fit into this mold, then it may be possible to combine those approaches with the techniques of programming language semantics and of logic and model theories, to construct complex and interacting accounts of ψ and of φ. To take just one example, the techniques that are used to construct mathematical accounts of environments and continuations might be brought to bear on the issue of dealing with the complex circumstances involving discourse models, theories of focus

in dealing with anaphora, and so on; both cases involve an attempt to construct a recursively specifiable account of nonlocal interactions among disparate fragments of a composite text. But the contributions can proceed in the other direction as well: even from a very simple application of this framework to this circumstance of Lisp, for example, I have been able to show how an accepted computational notion fails to cohere with our attributed linguistically based understanding, involving us in a major reconstruction of Lisp's foundations. The similarities are striking.

My claim, in sum, is that similar phenomena occur in programming languages and natural languages, and that each discipline could benefit from the semantical techniques developed in the other. Some examples of these similar phenomena will help to motivate this view. The first is the issue of the appropriate use of noun phrases: as well as employing a noun phrase in a standard extensional position, natural language semantics has concerned itself with more difficult cases such as *intensional contexts* (as in the underlined designator in *I didn't know that The Big Apple was an island*, where the co-designating term 'New York' cannot be substituted without changing the meaning), the so-called *attributive/referential* distinction of Donellan[19] (the difference, roughly, between using a noun phrase like "the man with a martini" to inform you that someone is drinking a martini, as opposed to a situation where one uses the hearer's belief or assumption that someone is drinking a martini to refer to him), and so on. Another example different from either of these is provided by the underlined term in *For the next 20 years let's restrict the President's salary to* $20,000, on the reading in which after Reagan is defeated he is allowed to earn as much as he pleases, but his successor comes under the constraint. The analogous computational cases include for example the use of an expression like (the formal analog of) *make the sixth array element be 10* (i.e., A(6) := 10),

19. Donnellan (1966).

where we mean not that the current sixth element should be 10 (the current sixth array element might at the moment be 9, and 9 cannot be 10), but rather that we would like the description "the sixth array element" to refer to 10 (so-called "L-values," analogous to MacLisp's SETF construct). Or, to take a different case, suppose we say "set X to the sixth array element" (i.e., X := A(6)), where we mean not that X should be set to the *current* sixth array clement, but that it should *always* be equal to that element (stated computationally this might be phrased as saying that X should "track" A(6); stated linguistically we might say that X should *mean* "the sixth array element"). Although this is not a standard type of assignment, the new constraint languages provide exactly such facilities, and macros (classic computational intensional operators) can be used in more traditional languages for such purposes. Or, for a final example, consider the standard declaration: INTEGER X, in which the term 'X' refers neither to the variable *itself* (variables are *variables*, not numbers), nor to its current designation, but rather to whatever will satisfy the description "the value of X" at any point in the course of a computation. All in all, we cannot ignore the attempt on the computationalists' part to provide complex mechanisms so strikingly similar to the complex ways we use noun phrases in English.

A very different sort of linguistic phenomenon that occurs in both programming languages and in natural language is what we might call "premature exits": cases where the processing of a local fragment *aborts* the standard interpretation of an encompassing discourse. If for example I say to you *I was walking down the street that leads to the house that Mary's aunt used to ... oh, forget it; I was taking a walk*, then the fragment "forget it" must be used to discard the analysis of some amount of the previous sentence. The grammatical structure of the subsequent phrase determines how much has been discarded, of course; the sentence would still be comprehensible if the

phrase "an old house I like" followed the "forget it." We are not accustomed to semantical theories that deal with phenomena like this, of course, but it is clear that any serious attempt to model real language understanding will have to face them. My present point is merely that continuations[20] enable computational formalisms to deal exactly with the computational analogs of this: so-called *escape operators* such as MacLisp's THROW and CATCH and QUIT.

In addition, a full semantics of language will want to deal with such sentences as *If by 'flustrated' you mean what I think, then she was certainly flustrated*. The proper treatment of the first clause in this sentence will presumably involve lots of "ψ"-sorts of considerations: its contribution to the remainder of the sentence has more to do with the mental states of speaker and hearer than with the world being described by the presumed conversation. Once again, the overarching computational hypothesis suggests that the way these psychological effects must be modelled is in terms of alterations in the state of an internal process running over a field of computational structures,

As well as these specific examples, a couple of more general morals can be drawn, important in that they speak directly to styles of practice that we see in the literature. The first concerns the suggestion, apparently of some currency, that we reject the notion of logical form, and "do semantics directly" in a computational model On my account this is a mistake, pure and simple: to buy into the computational framework is to believe that the ingredients in any computational process are inherently linguistic, in need of interpretation. Thus they too will need semantics; the internalisation of English into a computer (θ) is a translation relationship (in the sense of preserving φ, presumably)—even if it is wildly contextual, and even if the internal language is very different in structure from the structure of English. It has sometimes been informally sug-

20. See note 10, above.

gested, in an analogous vein, that Montague semantics cannot be taken seriously computationally, because the models that Montague proposes are "too big"—how could you possibly carry these infinite functions around in your head, we are asked to wonder. But of course this argument commits a use/mention mistake: the only valid computational reading of Montague would mean that mentalese (*S*) would consist of *designators* of the functions Montague proposes, and those designators can of course be a few short formulae,

It is another consequence of the view I am presenting that any semanticist who proposes some kind of "mental structure" in his or her account of language is committed to providing an interpretation of that structure. Consider for example a proposal that posits a notion of "focus" for a discourse fragment. Such a focus might be viewed as a (possibly abstract) entity in the world, or as a element of computational structure playing such-and-such role in the behavioral model of language understanding. It might seem that these are alternative accounts: what I am arguing is that an interpretation of the latter must give it a designation (φ); thus there would be a computational structure (being biased, I will call it the *focus-designator*), and a designation (that I will call the *focus-itself*). The complete account of focus would have to specify both of these (either directly, or else by relying on the generic declarative semantics to mediate between them), and also tell a story about how the focus-designator plays a causal role (ψ) in engendering the proper behavior in the computational model of language understanding.

There is one final problem to be considered: what it is to design an internal formalism *S* (the task, we may presume, of anyone designing a knowledge representation language). Since, on my view, we must have a semantics, we have the option either of having the semantics informally described (or, even worse, tacitly assumed), or else we ean present an explicit ac-

count, either by defining such a story ourselves or by borrowing from someone else. If the Lisp case can be taken as suggestive, a purely declarative model theory will be inadequate to handle the sorts of computational interactions that programming languages have required (and there is no *a priori* reason to assume that successful computational models for natural language will be found that are *simpler* than the programming languages the community has found necessary for the modest sorts of tasks computers are presently able to perform). However it is also reasonable to expect that no direct analogue to programming language semantics will suffice, since they have to date been so concerned with purely procedural (behavioral) consequence. It seems at least reasonable to suppose that a general interpretation function, of the Σ sort mentioned earlier, may be required.

Consider for example the KL-ONE language presented by Brachman et al.[21] Although no semantics for KL-ONE has been presented, either procedural or declarative, its proponents have worked both in investigating the θ-semantics (how to translate English into KL-ONE), and in developing an informal account of the procedural aspects. Curiously, recent directions in that project would suggest that its authors expect to be able to provide a "declarative-only" account of KL-ONE semantics (i.e., expect to be able to present an account of φ independent of ψ), in spite of the foregoing remarks. My only comment is to remark that *independence* of procedural consequence is not a pre-requisite to an adequate semantics; the two can be recursively specifiable together; thus this apparent position is stronger than formally necessary—which makes it perhaps of considerable interest.

In sum, I claim that any semantical account of *either* natural language or computational language must specify θ, ψ, and φ; if any are left out, the account is not complete. I deny, furthermore, that there is any fundamental distinction to be

21. Brachman (1979).

drawn between so-called procedural languages (of which Lisp is the paradigmatic example in AI) and other more declarative languages (encodings of logic, or representation languages). I A30 deny as well, contrary to at least some popular belief, the view that a mathematically well-specified semantics for a candidate "mentalese" must be satisfied by giving an independently specified *declarative* semantics (as would be possible for an encoding of logic, for example). The designers of KRL,[22] for example, for principled reasons denied the possibility of giving a semantics independent of the procedures in which the KRL structures participated; my simple account of Lisp has at least suggested that such an approach could be pursued on a mathematically sound footing. Note however, in spite of my endorsement of what might be called a *procedural semantics*, that this in no way frees one from giving a declarative semantics as well; *procedural semantics* and *declarative semantics* are two pieces of a total story; they are not alternatives.

22. Bobrow and Winograd (1977).

Annotations†

A1 ·1/20 «...Intro or whatever; maybe publication data goes here too?...»

«...Also say that this, too is a little simplistic?»

«...More serious is what I make of it; the Δ between this and external" and "conceptual role" semantics would be good to explain if I can ...»

«...Cf. A8 and A17 and A20...Also (p. 3) "use computational models to explain natural language semantics, if the computational models are themselves in need of semantical analysis"—this should be indicated (early) as a driving motivation...»

«...Note ·14/-1:15/0. This, as all the other papers in part C, emerge as worthwhile only in their later passages ;-)...»

A2 ·1/-1/1:3 There are three claims embedded in this sentence: (i) that being semantic is *a* distinguishing characteristic of computing; (ii) that it is *the* distinguishing mark; and (iii) that the necessary (whether sufficient or not) semantics must be *attributed*—where 'attributed,' as usual, was taken to mean *only* attributed; authentic or original semantics being the implied contrast. I certainly believed (i) in 1982, the year this was written; I would still believe it today, except for the fact, as discussed in "The Foundations of Computing" (ch. 1), that I no longer believe that computation is a theoretically interesting subject matter, and so would be reluctant to say that it has any disintuishing marks at all. Whether I believed (ii) or (iii) in 1982 I no longer distinctly remember.[1]

Even at the time, though, I was suspicious of a related thesis, then equally prevalent and still widely believed: that the formality condition—the claim that a symbol system works indepedently of the semantics of its ingredient symbols—is true of real-world computers. That the two claims are different is evident from the proposal, widely believed in the 1980s, in the heyday of classical AI and cognitive science, that the human mind might be a formal symbol manipulation system. If that were true, it would block any implication from formality to attributed semantics. One might imagine the converse implication to be more secure, on the grounds that if

†References are in the form page/paragraph/line; with ranges (of any type) indicated as x:y. For details see the explanation on p.·....

1. This paper was published earlier than any other in this collection, except for the fragments of the dissertation included in ch. 2. See annotations A17 and A20, below.

semantics are attributed, they must be relational, and hence would surely be metaphysically barred from playing a constitutive role in how computer work. But even in the late 1970s, as a graduate student, I was troubled by the ontological implications of this claim. Some form of semantics, it seemed to me, whether authentic or derivative or anything else,[2] was surely necessary in order to call anything *symbol* manipulation. If computation was defined as formal symbol manipulation, then the whole subject matter of computing might be relationally defined, making constitutive independence claims much more difficult to assess, especially ontologically. (Cf. also ·2/0/-5:-1, and annotation A6.)

A3 ·2/0/3 Instead of "linguistic facilities" I should have said *intentional* facilities (or capacities)—but cf. annotation A8, below. Note, too, that the term 'language' was (and is still) used in computer science as a general term covering representational systems than in linguistics or philosophy—without any implication of a system used for *communication*.

A4 ·2/0/5:8 The wording of this sentence betrays its early provenance.[2·5] What is evident is that we semantically interpret the function and display and buttons and other aspects of the *external* mereology of a calculator. But it is facile blithely to claim that we similarly interpret its *ingredients*. It may in fact be true—e.g., for the engineers and programmers—but most of us do not think about a calculator's ingredients at all (at least in any sense in which that is taken to mean "inner constitutive parts").

In general, at the time, I was inappropriately clear on the personal/subpersonal distinction in the human case, and on its machine (system/subsystem) analogue.

A5 ·2/0/-9:-5 Cf. the discussion of logic, and its constitution in terms of causal mechanisms that honour deferential semantical norms, in §… of the Introduction.

A6 ·2/0/-5:-1 Re the word 'attributed,' cf. annotation A2, above. My then-growing conviction that computing would never succumb to physical reduction (notwithstanding fn. 3), and the resulting implication that a theory of computing must rest on a theory of intentionality (rather

2.Cf. the discussion at … of how attributed or derivative semantics is still a version of semanticity; derivative does not mean *ersatz*.

2.5.Note that the paper was written five years before the publication of Dennett's *Intentional Stance*, the terminology of which could have been used here to good effect.

than the other way around), were the sorts of consideration that undermined my trust in the formality condition (to say nothing of challenging the very idea of "naturalizing" intentionality). I would now formulate this concern in terms of a rejection of blanket mechanism (Introduction, §…).

A7 :2/n4/3:5 The emotional tenor of the phrase "dashing the hope that computational psychology will offer a release from the semantic irreducibility of previous accounts of human cognition" betrays the fact that, in spite of the remarks above (cf. annotations A2 and A6), I was not yet (in 1982) at ease with abandoning what many philosophers would take as the prospects of naturalizing computation, let alone mind.[3]

A8 :3/1/2 I would now say—and would probably then have said—fundamentally *intentional* rather than fundamentally *linguistic*—but the paper was presented at a meeting of the Association for Computational Linguistics, and it was to that linguistic audience that the plea of "common subject matter" was being addressed.

A9 :3/1/4 The parenthetical indicates that I had doubts about the legitimacy of assuming that semantical theories should be formal—but the approach articulated in the paper (in terms of θ, φ, and ψ) would by most people's lights count as formal. The focus of my concerns with formality were (and have remained) primarily ontological, on whether computation itself was or is a formal phenomenon. I was less exercised about formal *theories*, where the predicate is assigned to the theoretical machinery used to analyze the phenomenon in question. Cf. the discussion of formality in §… of "The Foundations of Computation" (ch. 1).

A10 :3/1/-4:-1 Two subtleties lie below the surface.

First, those who use computational models to explain natural language semantics might feel that their semantical debts are ultimately discharged by mathematicians and computer scientists who provide semantical analyses of the computational systems and formalisms they employ in their analyses. Part of my brief in investigating semantics in a computational setting was to suggest that, even if such semantical accounts were to have been given (unlikely, as it happens—providing formal semantics for langauges that are used

3. Few philosophers would think that computation *needs* naturalizing, of course. For whatever reason—perhaps influenced by computing's status as a *science*—they would take it to be naturalistically palatable from the get-go. Clearly, even in the early 1980s I was already doubting that this was so.

is in fact rare), the concerns of such accounts, crafted by computer scientists, were almost sure to have been entirely behavioral, in spite of their use of seemingly referential semantical vocabulary,[4] and so the debt was unlikely to have been repaid. To answer to the legitimate concerns of linguistic semantics, in my view, would require a radically different *kind* of semantical analyses of computational systems than computer science took itself to be providing (as argued in ch. 2)—presumably along the lines suggested in this paper.

Second, someone might object that *all* forms of semantical analysis ultimately amount to no more than translation—since analyses are inevitably conveyed in language of some sort. **«—Yikes; fix the following—»** Fair enough; and this is not the place to take on metaphilosophical considerations of what it is to analyze, in English, the semantics of English (cf. debates about the adequacy of deflationary accounts of truth). But providing an analysis in English or another natural language, which it is reasonable to suppose is antecedently understood, is different from translating into a formal or computational system that cannot be accorded any such *a priori* (or perhaps even *a posteriori*) status.

A11 :3/-1/5:9 This discussion of the "operational cast" that has been given to the term 'semantics' in computing was written before I had formulated the distinction between *ingredient*ial and *specificational* views of programs—as for example discussed in "…" (ch. …); see also figure … of ch. 7.[5]

A12 :4//1:2 This was intended to be parsed as "(standard (model theories))"—i.e., model-theoretic analyses of the standard sort—not as "((standard model) theories)," in a sense that would mean theories of a or the standard model.

A13 :4//12:14 Cf. the discussion of compositional semantics in §… of the Introduction.

A14 :4//-8:-7 The "computational hypothesis of mind" would more normally be called the "computational *theory* of mind" (CTOM). Whether it is true that computational semantics presumes the CTOM could be debated, however. Likely most practicioners would make such an assumption—and especially in 1982 this was effectively a field-wide presupposition.

4. And in spite of any potential protestations on the part of the analysts.

5. A distinction that I have since come not so much to disparage as to blur. Cf. the dicsussion of the fan calculus in …

A15 ·3/n6/1: ·4/n0/4 Cf. the discussion of overlapping technical vocabulary in §... of the Introduction.

A16 ·5/2/4 «Cf ...»

A17 ·5/-2/-3:-1 (re fig. 2) This framework was developed for 3Lisp, as discussed in Part B. Note that this paper was presented in 1982, the same year that the 3Lisp dissertation was completed. The papers presenting 3Lisp, included here as chs. 3 and 4, were not presented until two years later, in 1984 (though I had talked about it widely; see annotation A20, below).

A18 ·6/0/3:4 Calling the function 'θ' mapping external expressions ("notations") onto internal structures an *interpretation* function was a passing usage. It is not how I generally used the term 'interpretation'; in fact by the end of the paragraph I have noted that I henceforth called it 'internalization." Using the term 'interpretation' for this relation will seem odd to logicians and philosophers—and to most computer scientists as well. It was so common at the time, however, and to some extent still is today, for researchers in AI and cognitive science to refer to the internal correlates of natural language expressions as their "semantics"—and to talk about "computing the semantics" of natural language (cf. annotation A... in ch. ...)—that I may have introduced the relation in this way here in order to mesh with that practice. (Cf. ·9/1:·10/1.)

Throughout, however, my primary concern was to distinguish that relation from what I took to the more substantive semantical relation 'φ' relating "expressions in $\mathcal{S}$ to the world of discourse.[5.5]

A19 ·8/0/12:13 This is the "antisemantical" reading of formality discussed in §... of "The Foundations of Computing" (ch. 1), which I would soon come to challenge. (See also annotations A... in ch. ..., A... in ch. ..., and A... in ch. ....)

A20 ·8//-2:-1 This is a reference to the 2Lisp and 3Lisp work discussed in Part B. Cf. annotation A17, above, as regards dates. This paper was presented before any papers on 3Lisp had been written, though during 1982 I

.5.5. Note (in spite of ·7/0) that to assume that 'θ' deals with relations between external language and internal structures, and 'φ' with relations between internal structures and external task domains, is to assume that the linguistic-expression/symbol/referent and outside/inside/outside boundaries align—which may often be true, but is certainly neither always nor necessarily the case, even if by definition the "internal elements" are always inside mind or machine. Cf. §... of ch. 1, and much of Part B

had given more than a dozen talks on 3Lisp and reflection, one of which triggered SRI International's Jane Robinson to request that I prepare this paper for the 1982 meeting of the Association of Computational Linguistics.

A21 ·9//-3:-1 As I soon came to realize, indexicality and deixis are in no way so easily dismissed. Cf. 03 and my (forthcoming) "Who's on Third?", where I argue that deixis is a foundational fact about the universe—in effect, being the underlying ontological condition that warrants the fact that differential equations are the preferred epistemic way to formulate its fundamental physical laws.

A22 ·10/2/7 The word "independent" is too strong. One of the points of the 3Lisp analysis in Part B is to show that ψ and φ can be theorized only together, in terms of an overarching "full significance" function Σ. What I meant in this passage is that I had not seen an analysis of a programming language that gave accounts of *both* ψ and φ, recognizing their conceptual difference, even if they were inextricably (and normatively) interlinked. Cf. the discussion of logic in §... of the Introduction (and annotations ... in ch. ... and A... in ch. ...).

A23 ·10//-5:-1 Strict use/mention hygiene requires keeping these apart: program identifiers that designate data structures, on the one hand, and the data structures thereby designated, on the other (and both from the people and money and such that the data structures in turn designate or carry information about). Nevertheless, this is an example of the sort of semantical strictness that I have come to believe is theoretically untenable—i.e., that *always* maintaining the distinction complicates theory to the point of obfuscating the fundamental regularities that require explication, and is ontologically unjustified and unjustifiable as well. Cf. "The Correspondence Continuum" (ch. 10) and "The Foundations of Computing" (ch.1).

Providing a flexible way of making such distinctions only if and when appropriate is one of the goals of the fan calculus (cf. «...»).

A24 ·11/0/-4:-1 Cf. annotation A11, above. That terms (e.g., identifiers) in programs are meta-syntactic (more accurately: meta-structural) is one of the insights that lead to making the distinction between *specificational* and *ingrediential* view of programs discussed at length in ch. 2.

A25 ·12/2/-2:-1 Re Teco and Smalltalk see annotations A... in ch. ... and A... in ch. ..., respectively.

A26 ·12/3/1:2 This is another endorsement of the then-ubiquitous view that com-

puters are formal symbol manipulators, in the antisemantical sense of 'formal.' Cf. annotation A19, above.

A27 ·12/3/4:-1 For example, the effect of running or executing the program fragment "x := f(y) + g(z)" might not be to set variable x to anything, if, while executing g(z), the body of that procedure were to perform a non-local exit from the entire encompasing procedure, or were otherwise to invoke continuations (or some other similar construct) so as to violate procedural locality.

A28 ·13/0/-3:-1 The phrase "that *humans* can interpret" is an implicit reference to the presumption that the semantics of a formalism are inevitably attributed; cf. annotations A2 and A6, above. The main point, though, is merely that a formalism must have a denotational or declarative interpretation (φ) in order to count as a *logic*; cf. the discussion in §... of the Introduction. Nothing should be inferred from the use of the word 'interpret' here that the (attributed or not) semantics would need to be *intelligible*.[6]

A29 ·14/0/3:4 In fact one cannot actually *say* anything in Prolog either, its alleged declarative semantics notwithstanding. In spite of what is said in the rest of this paragraph, expressions in a Prolog program do not have assertional force.

A30 ·19//-2: ·20//3 This statement (that there is no difference between procedural and declarative languages) is a little glib. Nor is it clear, as the passage would suggest, that the two varieties exhaust the space of possibilities. Cf. the preceding annotation about assertional force, for one thing; it is not clear, at least in any systems we would presently consider paradigmatic, that one can assert anything in a calculus of either type. My point was only to deflect the idea, common at least at the time, that an adequate semantical analysis of procedural languages could limit itself to focusing on effect and behavioral import (ψ)—and of declarative languages, to issues of reference or denotation (φ). As indicated by its last sentence, the brunt of the paper

6. It is not uncommon to think that connectionist networks and neural models of cognition (and brain) are not "symbol manipulation systems," not only in the snese of not containing discrete, compositionally interpretable symbols, but more strongly as not being representational or semantic at all. But the fact that their builders do not so interpret them does not mean that they are not semantic, of course. The more interesting hypothesis is that, sure enough, they are or anyway may be semantic—but that their interpretations (φ) are or may not be humanly intelligible.

is to argue that both types of calculus, whatever their differences, require both forms of analysis—normatively tied together.