Annotations[1]

A1 ·1/1/-8 Apologies for the sexist language (here and throughout). In spite of the awkwardness of the last word, I would now say 'their story applies to themself'—see fn. … in the Introduction.

A2 ·1/1/-4:-1 The theme of focus requiring a modicum of distance—that just as one cannot see what is too far away, so too one cannot see that which is pressed right up against one's eyeballs—is explored in *On the Origin of Objects* (03). As mentioned in ch. 2, a number of metaphyical points are reflected in the design of 2/3Lisp which might not strike one's attention, if one were to focus on them solely from the programming language perspective—or perhaps to put it more accurately, if one's perspective on programming languages is to view them as their own distinct genus, rather than being (as I have always assumed) an instructive species of languages and associated intentional processes more generally.

A3 ·2//1 As noted in annotation A…, throughout this book I have deleted the hyphen in the names of the dialects of Lisp under discussion: hence '3Lisp' instead of the '3-Lisp' of the published version.

A4 ·2/-1/-7:-4 This phrasing is somewhat disingenuous—that "[t]he point is not to decide at the outset what should and what should not be explicit." In a procedurally reflective dialect of the sort presented here, the language *designer* must decide, in advance, what aspects of the language or system will potentially be available for explicit treatment in user code. Any such aspects must be explicitly dealt with in the metatheory in terms of which the processor and dialect are defined, and then provided for in the resulting implementation. The point is that the language's *users*—i.e., programmers—need not decide in advance which of those aspects to avail themselves of in any given program. More significantly, even within the context of a single program, the relevant factors need be dealt with only in places where their explicit treatment is germane; otherwise they can be left implicit.

The paper would thus have been better phrased if the sentence had been written as follows:

> "Although the metatheory (and reflective processor) must deal explicitly with all of those aspects of the language that

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

> are capable, at any point within a program, of being made explicit, user code that does not want to deal with them need not deal with them explicitly. In Steele's dialect, in contrast, for an aspect to be treated explicitly at any one point in a computation, it must be dealt with explicitly throughout the whole program. In a sense, therefore, reflection can be understood as providing something like *contextual information hiding*—or perhaps more accurately, *contextually-dependent explicitization of otherwise implicit information*."

The next sentence in the text is more accurate, and more useful.

A5 ·3/1/1:3 Italics have been added in this version. As noted in ch. 2, it is this thesis that went unnoticed (or at least was set aside) when this paper was first published.

A6 ·3/1/4:6 It is here where the overarching deferential perspective on logic, representation, etc., enters the picture—as discussed in §5.c of the Introduction, and extensively in ch. 2. The framework in terms of which 2Lisp and 3Lisp are defined is based on a theoretical formulation of this "semantic significance that transcends...behavioral import." Specifically, the significance that "transcends" is analysed in terms of declarative import (called 'ϕ', for 'philosophy,' analogous to semantic interpretation or denotation in logic), which establishes the norms that govern behavioural import (called 'ψ', for 'pscyhology,' analogous to reasoning, derivation, or proof theory in logic).

A7 ·3/1/-7:-5 See §3, particularly the first full paragraph on p. 10, but in truth this paper does not provide an adequate analysis of the notions of operational and denotational semantics. See however ch. 2, ch.10 (especially its annotation A...), and also Volume III of AOS.

A8 ·3/-1/4:5 "[T]he general intellectual hygiene of avoiding use/mention errors" reflects the position discussed in §6 of the Introduction of embracing a traditionally strict semantical framework of the sort embraced in formal logic. Hygiene, however, is theory-relative. See "The Correspondence Continuum" (ch.11), written approximately five years later, by which time these commitments had already begun to unravel.

I still believe that the 2/3Lisp architecture has a degree of theoretical elegance. The problem is just that this elegance proves un-

workable. As briefly discussed in the Introduction, one aim of the fan calculus is for it to provide a theoretical framework that accommodates the kinds of semantical flexibility described in ch.11, so that the style of theoretical cleanliness illustrated in 2/3Lisp can be exemplified in a radically more practicable architecture.

A9 ·3:-1/-1:·4/0/-1 By 'semantic rationalization' I mean defining the processing regimen to "normalise" in a "reference-preserving" way. Of the two properties, it is the latter notion, of *reference-preservation*, that is crucial—endowing the dialect with what in chapter 1 of the dissertation (ch. 3b, at ·...) I refer to as *semantic flatness*.[2] The former property, of *normalisation,* is both more restrictive and less important—i.e., the requirement that the co-referring result be contextually-independent, side-effect free, and stable (§4, ·...; see also ch. 3b, ...). As discussed in ch. 2, at ·..., this particular construal of normalisation is essential to the particulars of 2Lisp and 3Lisp, but it is not a condition on the overarching notion of a procedurally reflective architecture being proposed.

The entire discussion, however, rests on the much more important underlying point, explored in ch. 2, that unless one distinguishes *what happens* in a system from *what it is about*—i.e., distinguishes *reference* from *normalisation* in the 2/3Lisp case (what later in the paper I refer to as φ and ψ)[3]—then one cannot even claim that the system is reflective at all. While one might challenge the thesis that reference preservation (semantic flatness) is a necessary or useful architectural criterion in terms of which to define an intelligible reflective system, that is, a strong version of the approach being defended in this paper is that it would be incoherent to challenge the prior act of distinguishing φ and ψ. Without making that discrimination, it is claimed, the notion of reflection cannot even be *defined*.

This is why it is critical for 2/3Lisp to adopt an *ingrediential* rather than *specificational* view of programs and their semantics.[4] The fundamental characterization of a reflective system is of

2. Cf. also §4's characterization (p. 21, ¶0) of traditional Lisp evaluators "crossing semantic levels." Note that the use of the term 'semantical' is inappropriate in this situation; the phrase should be 'crosses semantic levels'; cf. fn · in the Introduction.

3. Defined in §3; see especially ·....

4. See §3, particularly ·....

a system that reasons *about* its own operations and structures and behavior, where the applicable notion of "aboutness" must be different from what it is that happens. Cf. for example the following phrasings in §2 of the paper (emphasis added):

> ·5/1/4:6 "not only…*about* its self and internal thought processes, but also…*about* its behavior and situation in the world"
>
> ·5/-1/-6 "*refer to* or *deal with* other parts of a computational system"
>
> ·6/1/4:5 "reasoning directly *about the world* and reasoning *about that reasoning"*

and similar passages in §3 and §4. If one adopts a specificational view, and takes a program to denote the behavior that results from executing it, then in order to define that program as reflective one would have to define two distinct notions of designation or aboutness. As I indicate in «ref», various such strategies suggest themselves, such distinguishing the semantics of programming *languages* from the semantics of programs *written in those languages*, or—perhaps more productively—distinguishing *program semantics* from *process semantics* (by the latter meaning the semantics of the processes that result from running executing the programs—what on a specificational view of programs could be desibed as "the semantics of the semantics of the programs"). See ch. 2, :…, ch. 11, and ch. 12.

A10 ·4/2 It would have been helpful to point out that while it is possible to understand recursion (mathematically or computationally) without explicitly distinguishing declarative import (φ) and procedural consequence (ψ), the same does not hold for reflection. The notion of reflection is only intelligible with respect to a disentangled understanding of both.

A11 ·5/1/2:6 On the characterization given here, which is compatible with the formulation in ch. 6, 3Lisp was in fact only *introspective*, not fully *reflective*. As this passage illustrates, however, my intent had always been to aim for the wider notion—as evident, for example, in the encompassing project of defining the general-purpose representation system Mantiq, of which, as described in §3 of the Introduction and in ch. 2, 3Lisp was originally intended to be a design study.

A12 ·6/0/-3:-1 In part this is a reference to Mantiq (see previous annotation), but as mentioned in ch. 2 I had also planned to develop a next dialect in the series of reconstructed dialects of Lisp, to be called 4Lisp, which, while otherwise retaining 3Lisp's basic style and control structure, was to include semantically-rationalized data structures for (external) reference to the real-world. That is: 4Lisp was intended to extend 3Lisp's φ/ψ semantic framework to data structures as well as programs. As noted in ch. 2, 4Lisp, like Mantiq, never materialized, due to the challenges of developing representational regimens adequate to real-world ontology.

At the "Reflection '96" workshop, held April 21-23, 1996 in San Francisco, I presented a paper entitled "What Ever Happened to 4Lisp?"—which remains unpublished. Fundamentally, as described in ch. 1, the issues were both semantical and ontological. The semantical issues included not just confusions and ambiguities about programming languages, programs written in them, processes, etc., of the sort discussed here, but the more challenging issues expressed in ch. 12. But it was the ontological issues that were the most daunting. One way to understand *On the Origin of Objects*, published that same year, is to view it as an initial exploration of how would have to understand the world in order to do justice to 4Lisp's and Mantiq's goal of being genuinely reflective, rather than merely introspective.

A13·6/1/10:11 The two connections are reminiscent of what philosophers describe as *word-to-world* (or *mind-to-world*) and *world-to-word* (or *world-to-mind*) "directions of fit,"[6] with the following exceptions:

1. Instead of words or minds we are of course talking about internal structures (impressions) in a computational process;
2. The issue is not just that the "words" or internal structures should end up true or "fitting" what is the case—by adjusting the "word" in the world-to-word case, and adjusting the world in the word-to-world case—but rather that the connection be causally efficient, so that that "fit" can be practially accom-

6. See for example Austin (1962) and Searle (1979).

7. Note that internalization (θ) and externalization (θ^{-1}) are both effective in 2/3Lisp, ensuring that language is included along with structure in both directions of causal connection.

7.5. E.g., see ··· in ch. 12, and the discussion at ···9/1 of ch. 11.

plished;[7] and

3. The "world" at stake is the internals of the computational process itself, as discussed in ch. 2. As usual, 3Lisp's design was meant to exemplify semantical and metaphysical frameworks of much wider applicability.

A14 ·7/1/7:9 To say that implementations are "really...just descriptions" is glib, as are the next two sentences. As discussed in ch. 2, the relationship between description and implementation is a something like a pun or stalking horse that permeates not only this whole paper, but the entire reflective architecture and background theoretical perspective under discussion throughout this volume.

As noted in ch.2 (see §·...), the focus here on implementation-inspired examples (stack frames, processor state, etc.)—not just in this and the subsequent paragraph, but to an extent throughout the chapter—was in retrospect unfortunate. What mattered was to get at aspects of a running process that were implicit, from the point of view of the program; for a paper aimed at a community of programmers and computer scientists, I thought that these implementation-oriented examples would be compelling because familiar. The difficulty was that they ended up conveying the misimpression that reflection *has to do with making the implementation explicit and available*—not my intent at all, and as argued in ch. 2, a considerable distraction from understanding what reflection is really about. See in particular ch. 2, §...

A15 ·8/-1/1:2 Elsewhere I refer to internal structures as **impressions**, to highlight the contradst with expressions.[7.5] As soon as the term 'impression' was introduced, it took over from 'structure' in conversation among those working with 2/3Lisp—including Jim des Rivières (co-author of ch. 5) and others in my Xerox PARC research group, and more recently Jun Luo and other students and colleagues.

A16 ·9/n3/3:6 In March of 1982, a month after the dissertation had been completed, and still a rather green graduate student, I presented 2Lisp and 3Lisp at a colloquium at SRI International. To my astonishment, the legendary John McCarthy, inventor of Lisp, sat down at the very front of the room. I was terrified. There being nothing for it, I presented my analysis of traditional Lisps, including the critique of evaluation given in §4, and hazarded the idea, stated in this footnote,

8. Quoted function descriptions could be passed "downwards" because vari-

that McCarthy had recruited (structural) quotation and dynamic scoping to compensate for the fact that Lisp 1.5 was (inadvertently?) only first-order. Guy Steele and Gerry Sussman had already developed the higher-order, lexically scoped Scheme, and presented their analysis of Lisp in the now-famous *Lambda Papers* (Steele & Sussman, 1975–80). So the idea of a lexically scoped, genuinely higher-order Lisp was not new; it was the analysis of reference and quotation on which I did not know McCarthy's views. While hugely impressed by Steele and Sussman's papers—indeed their work on Scheme undoubtedly influenced my choice of Lisp as a site for a Mantiq design study (up until that point I had been working in the area of knowledge representation, not programming languages)—I had also felt that they did not go nearly far enough. Scheme discourages (as we might now say, "deprecates") quotation, whereas I felt that there was extraordinary power latent in quotation, which needed to be theoretically understood, rigorously disciplined, and then unleashed.

I have no reason to believe that McCarthy ever thought much about (or of) 3Lisp after that talk, but I was hugely relieved, after I made the claim that Lisp 1.5, within the context of a dynamically scoped environment, had recruited quotation in order to "fake" higher-order behavior (at least downward[8]), that he agreed—nodding his head slowly, and saying (I believe I remember the words exactly) "I am sure that is right."

A17 ·10/1/5:7 When this was written I still believed that with sufficient rigorous attention these complexities could be sorted out—cf. for example ch. 11. By the time I had written ch. 12, however, I had begun to appreciate the magnitude—and ultimate formal incomprehensibility—of the task.

A18 ·11/0/7:8 Even at the time this paper was published I was critical of the idea that computation could adequately be understood as formal symbol manipulation. I believe that my use of the phrasing "in the sense that" was meant (rather ineffectively) to signal some distancing of

ables bound in the calling procedure would still be dynamically available for use by those descriptions if executed underneath the point of call on the stack. What did not work was to pass them "upwards." That is: quoted function descriptions functioned properly (within limits) as *arguments* to procedures; they did not as functional *results*.

8.5. See annotation A…, above.

my own view from that then-universal assumption. It was not until 1986 that I explicitly argued against such a construal. See "The Link from Symbols to Knowledge" (Smith 1986b) and AOS (especially Volume II).

A19 ·12/0/3 What I am here calling "structures" (data structures, programs in the sense that they might be available during the course of their execution, etc.—i.e., what I later called "impressions"[8.5]) are at least arguably *more* abstract than linguistic expressions, but that is not to say that they are *abstract objects*, in the sense enjoyed by genuine mathematical entities, types, etc.[8.7]

A20 ·12/1/6 The words 'the relationship to' has been added to the original, for clarity.

A21 ·12/1/-1 Cf. annotation «...» of ch. 3....

A22 ·14/0/2:3 In discussions of 2/3Lisp the term 'numeral' is *overloaded*, as computer scientists would say—used to refer both to (external) strings and to (internal) structures. Thus in this paper the (English) phrase "the numeral 3" is used to refer to an internal 2/3Lisp structure (impression). To refer to an external 2/3Lisp (string) numeral I would instead use "the numeral «3»'. See the fn. labeled '†' on p. 18.

A23 ·14/-1/-4 It would have helped if this had been written: "is always *normatively* related to semantics."

A24 ·14/-1/-2:·15/0/1 While correct as stated, this parenthetical would have been more perspicuously formulated as: "which, incidentally, can be expressed as the equation $\psi(s_1, s_2) \equiv \varphi(s_1) \subset \varphi(s_2)$, if one takes ψ to be a relation, and φ be a function mapping sentences onto possible worlds that satisfy them."

A25 ·15/0/-2:-1 TECO ("text editor & corrector") was a string-processing language which ran on the "Incompatible Time Sharing System" (ITS) at the MIT Artificial Intelligence Lab in the 1970s. It is now remembered primarily as the programming language in which the initial versions of the still-popular text editor EMACS were written. Smalltalk, an object-oriented, dynamically-typed, and inchoately "reflective" programming language, was developed at the Xerox Palo Alto Research Center (PARC) by Alan Kay and his colleagues during the 1970s.

A26 ·15/-1/4:5 Cf. ch. 11, ch. 12, §6 of the Introduction, ch. 2, and Volume III of

8.7. See aos for an argument that *all* objects are to *some* extent abstract.

8.8. See http://www.thocp.net/biographies/dijkstra_edsger.htm «... is this the source? ... »

9. See annotations A11 and A12, above (p. ·).

AOS.

A27 ·16/0/3 To say "primarily communicative" makes sense only on a view that language exists solely as a medium of interchange or interaction among minds (or other intentional systems or processes). On such a view, to say of a text that it is a "description," for example, would be understood as shorthand for saying that it can be used between and among intentional agents so as to enable *them* to stand in a descriptive relation to that which is described. In the philosophy of language or mind this would be viewed as a strongly psychological or mentalist view of intentional content-bearing. While in the 2/3Lisp context it is evident why I was distinguishing this case, to put it this way in the text is rather glib. In other places (e.g., in «ref») I sort views of programs into three rather than two types: *ingredi-ential*, *specificational*, and *communicative*.

A28 ·16/0/11 The POPL version contained the word 'avoid' instead of 'allow', but that was a mistake. The intent of the sentence is this: that "the only salient difference between specifications in general, and programs as a particular species of specification, is that in the general case specifications may use non-effective concepts in describing behavior (such as specifying a square root routine by saying *returns as output a number that, if squared, equals the input*), whereas programs must be effective."

A29 ·16/-1/-3:-1 From a 2014 perspecttive (30 years after this paper was written), I confess to having no idea of what the last sentence of this paragraph was intended to mean.

A30 ·17/1/3 Cf. §6 of the Introduction, and much of ch. 2.

A31 ·17/1/7:8 A general significance function (Σ) of this sort was defined in the dissertation; see the sections included in ch. 3.

A32 ·17/1/-7:-6 This statement ("the relationship to the world that s_1 signifies") is infelicitous. It is not the *relationship* to the world that the structure signifies; rather, it is in virtue of participation in such a relationship to (or with) the world that the structure is able to signify whatever in-the-world entity that it does.

A33 ·17/1/-2:-1 Computer science talks about a variable being "bound to" something—namely, to what is called its 'value'—though, as evident in the semantical reconstruction being carried out here, I take it that that usually means a co-referential structure. Strictly speaking, that is

(and *pace* the considerations adduced in ch. 12), a programming language variable would be bound to a *numeral*, not to a number—and should be so described in contexts in which the differences between numerals and numbers are significant. In mathematics and logic, variables are likely, if bound *to* anything, to be understood as bound to *numbers*—i.e., to what is here being called declarative import. Moreover—and this is what tripped up conversations between Barwise and me, it is more common in mathematical logic to describe a variable as "bound *by*" something—namely, by quantifiers, scoping constructs, etc. In computer science, that is, to say of a variable that it is 'bound,' *simpliciter*, is understood as an abbreviation for, or as implying, that it is "bound to (something)"; in logic, as an abbreviation for, or as implying, that it is "bound by (something)".

This is just one small instance of the general phenomenon of computer science's using, as technical terminology, vocabulary and phrasings derived from logic, but in its own distinct ways—an instance of the topic discussed in §4 of the Introduction. Sometimes, as here, the differences are subtle, and not usually distracting; sometimes, as with the word 'semantics,' they are major, and cause considerable confusion. See AOS.

A34 ·19/1/1 This section title is a play on Edsger W. Dijkstra's legendary "GO TO Statement Considered Harmful" (*Communications of the ACM*, Vol. 11, No. 3, March 1968, pp. 147–48). No computer scientist in the 1980s would have failed to recognize the illusion; the *Communications of the ACM* (Association for Computing Machinery) was the première professional computer science journal at the time, and Dijkstra's letter was widely taken to have inaugurated serious theoretical analysis of programming. Cf. this note from the History of Computing Project «ref»:[8.8]

> "In 1968 Edsger Dijkstra laid the foundation stone in the march towards creating structure in the domain of programming by writing, not a scholarly paper on the subject, but in-

10. A variant of vanilla λ-calculus extended with a special operator '✶', so that, modulo some subtleties, the term '✶' emerged as the result of normalizing any complex expression in which that form occurred.
11. Bobrow, des Rivières & Kiczales (1991).

stead a letter to the editor entitled "GO TO Statement Considered Harmful". (*Comm. ACM*, August 1968) The movement to develop reliable software was underway."

A35 ·21/0/1:4 From a restricted view of programming languages, it may seem fastidious to make heavy weather out of the difference between a number and a numeral. But the philosophical issues at stake are substantial. Note, to take a striking example, that in his 1985 Presidential Address to the American Philosophical Association, Fred Dretske used this very distinction not only to claim that calculators cannot add, but to go on, more seriously to argue hat computers cannot and never will be able to think—i.e., to undermine the very possibility of what Searle has famously dubbed "strong artificial intelligence." (Dretske 1985)

A36·23/-1/-2:-1 This is as close as this paper comes to an explicit statement of its underlying commitment to a *deferential semantics* (cf. §... of the Introduction, and ch. 2).

A37·27s/1/1:2 See annotation A16, above.

A38·28/2/-2:-1 It is this ruthless semantic strictness to which I was referring, in §1 of the Introduction, when I said that in designing 3Lisp I had achieved a degree of philosophical clarity but at a price of "unusably fastidious baroqueness." «...check!...»

A39 ·29/-1/-4 Which behavior, of course, they must also designate, as well as engender. In fact one might describe an MPP as a program P where $\psi(P) \approx \varphi(P)$.

A40·30/0/1:2 Talk of recursive definitions being "self-referential" was typical of the sorts of semantical sloppiness that first puzzled me, and then frustrated me, during the course of my computational education. It was this sort of conceptual inelegance that I was trying to rout from computational discourse through this exercise.

As mentioned in the Introduction (see esp. §6), I now believe this "sloppiness" to have two components, the disentangling of which will require considerable artistry. One component stems from a metaphysical/ontological fact of the highest order: the world is simply more complex than is or ever will be comprehended in an imaginable theoretical framework. Adequate accounts, therefore, in my judgment already do and increasingly have more in common with novels that many formalists may like to admit. What detail

is given what sorts of shrift is an act of judgment, not a matter of formulaic prescription. That, I believe, will always be true. That said, the second component of present-day sloppiness stems from the profound inadequacy of current theoretical frameworks to do justice to even what regularities there are, as regards the sorts of complexity adduced in ch. 12.

A41 ·31/1/-9:-1 Cf. ch. 12, especially §8, ·….

A42 ·33/0/-1 That we have no satisfying theory of process was a claim I had come to believe while a graduate student at MIT, and it is one I would still endorse. In the 1970s, though, I believed we were on firm ground with respect to ordinary objects—a view that by the 1990s I had clearly lost confidence in (hence *On the Origin of Objects*). A glimmer of the sorts of concern I felt is given in the discussion in §8 of the Introduction; see also ch. 1 §…, and AOS.

A43 ·35/1/1:6 This idea of there being a "reflective act" accords with what in §… of ch. 3b and in ch. 5 is called the "level shifting view," with the anima or active agency shifting up or down, in what logicians and philosophers might describe as cases of *semantic ascent* and *semantic descent* (as usual, the analysis is permeated with the "pun" as between description and implementation). The model described in the next sentence, in contrast, describes what the "tower view," in which no shifting is going on, and all levels are simultaneously active. See «…».

A44 ·43/n/-3:-1 See the discussion in ch. 5, especially of primitives at ·….

A45 ·44/0/-3:-1 This issue of striking a balance between vantage point and causal connection presages the notion of "partial disconnection" that is a major theme of *On the Origin of Objects*.

A46 ·44/1/4:5 RETFUN and FRETURN were "non-local" return functions in various 1970s-era Lisp dialects, which could be used to implement out-of-the-ordinary control structures—e.g., for complex error recovery. Current records suggest that FRETURN existed in MacLisp, Interlisp, Franz Lisp, and various other dialects of the day.

2Lisp and 3Lisp were first implemented in MacLisp, when I was a graduate student at the MIT Artificial Intelligence Laboratory,

12. Handles vanished, for example, so that 2, '2, ''2, '''2, etc., all turned into the same thing—namely, the *number* (not numeral!) two.

where it was developed and maintained. Subsequent (and better) implementations—including a just-in-time incremental compiler—were written (primarily by Jim des Rivières, co-author of ch. 5) in Interlisp at Xerox PARC, where I subsequently worked.

A47·45/0/-2:-1 The argument in this one-paragraph section, as well as being glib and abbreviated, is in fact unsound (it is too strong). The referenced Smith & des Rivières (1984)—"Implementation of Procedurally Reflective Languages—is included here as ch. 5; it presents a much better analysis of why 3Lisp is tractable, as well as a full implementation. The total program, including all utilities, was about 200 lines of 2Lisp; however, the "50 lines" referenced in the text is not entirely misleading, as that includes the substantive part (everything except essentially trivial subroutines).

The reference to "Smith forthcoming" is not detailed in the references in the POPL paper, and at present (2012) I have no recollection of what in particular I had in mind—or even whether I *did* have anything particular in mind. My guess is that I expected that someday I would work on such issues of tractability—including the indicated issue of whether an implementing processor could be algorithmically derived from an RPP. But just as in the case of a mathematical theory of reflection, none of this came to pass. Instead, my attention moved to the issues cited above about 4Lisp, reference to the external world, and genuine reflection (as opposed to the introspective abilities exhibited here).[9]

A48·45/1/-3:-1 In the mid-1980s, the reflective λ-calculus described in §6 of the Introduction[10] «...and elsewhere? check ...» was defined and presented to a logic colloquim, hosted by Jon Barwise, at Stanford's Center for the Study of Language and Information (CSLI). The presentation engendered reactions that I can only describe as ranging from befuddlement to consternation. There is no doubt that my intention of convincing Jon Barwise (a very good friend) that reflection was important, and how it worked, entirely failed.

«...Ref other places this is talked about...unify?...»

A49·46/2/1:8 See the discussion of LAMBDA and intensional procedures in what was §4.c.i of the dissertation on 3Lisp, included here as ch. 3c.

A50·46/1/-2:-1 As in the case of so many other ideas gestured towards in this paper, I myself did not end up pursuing this suggestion—for all of the reasons adumbrated above (semantical and ontological difficulties).

It is perhaps worth mentioning, however, that one of the design goals for Mantiq was to define a "structural field," along the lines of what is defined here (see 8/1/4:5 in §2) but at a sufficient level of abstraction so that *quotational* reference and *hyper-intensional* reference could be fused. The idea would be that at least one meaningful notion of "means the same thing" could be tested, in such an architecture, merely by checking structural identity.

A51:47/0/-5:-3 The prospect of defining a system in which one could, within any particular routine, or at relevant points within a program, reify only *those aspects of a computation that were relevant and required explicit treatment*, was an idea that I had wrestled with in the very early stages of the design of 3Lisp, while still at MIT, as was the idea of what it would be to define a higher-order "language design" library from which one could "load in" such modules as *recursion*, *reflection*, etc. Both ideas were bruited about in my research group at PARC throught the 1980s—although, needless to say, neither was brought to fruition in the form that I imagined them (in particular: in a way that incorporated the deferential semantics and distinction between declarative import and procedural consequence that as so fundamental to 2/3Lisp).

In particular, although the initial work on both *metaobject protocols* (MOP)[11], the Common Lisp Object System,[11.5] and *aspect-oriented programming* (AOP) emerged out of these discussions from members of my research group, I was struck that the semantical (referential) aspect of 3Lisp, the part that I felt was most foundational and theoretically important, was never incorporated into these developments. Public discussions of both notions invariably take a specificational view of programs, according to which the "meaning" or semantics of a program has to do with what behavior it engenders (and thus instantiate what in the Introduction I call a philosophy of *blanket mechanism*).

As a result, in my judgment, neither the MOP nor AOP frameworks have adequate intellectual machinery in terms of which even to define the notion of "meta." That is not to say that issues of reference are not at play; I do not believe it is possible to call one thing "meta" to another except in referential terms. Rather, the point is that the referential aspects, though utterly crucial, as yet remain wholly implicit (they remain, to put it in terms of §... of ch. 2, wholly within

the "tacit view of programmers").

A52 ·47/1/4 Re the reference to "Smith (forthcoming)," see the discussion in annotation A47, above. In 1984, however, I did set out on the project described in §... of ch. 2: working with Joseph Goguen and Jose Meseguer to develop a formal, mathematical account of the denotational semantics of 2Lisp. As recounted in «...», when they presented their proposal I was stunned. The reference relation (φ), which I had worked so hard to bring into view and honor, and which was so foundational to the architecture, had been entirely obliterated.[12] It was not just that effectivelyeverything that mattered to me about 2Lisp had disappeared. The real issue was that I was unable to explain to them *why* what had disappeared had mattered to me—why that which they took to be *eliminable,* en route to semantical cleanliness, was exactly what I took semantical cleanliness *to consist in*. That was what rocked me back on my heels, and led me into the foundational investigations that have continued to occupy me ever since.

A53 ·47/-1/7 For example, it would be essentially trivial to construct a λ-calculus analogue or mirror of the implementation presented in des Rivières and Smith (1984), included here as ch. 5.