Annotations[1]

A1 ·1/-1/5 It would also have been correct, and philosophically more expected, had this been written "adequate theories of *intentionality*"—i.e., theories of intentionality-with-a-t,[2] the full gamut of issues involved in how it is that a sentence or structure or event α can be meaningful or about something. I did however intend the more specific intensionality-with-an-s.

As the discussion throughout this chapter makes evident, and as is highlighted in dissertation §4c.i,[3] in any reflective system of the sort envisaged one needs to deal not only with *extensional* issues, having to do with the reference, denotation, or designation of symbols and other intentional entities, but also with usually finer-grained *intensional* notions of *what they mean*. As usual, the issue comes up not only in the human case, where reflection typically involves thinking about the intensional (meaningful) content of other thoughts and epistemic states, but also in computational contexts (see for example ·86/1 and ·108/1).

In many respects the architecture presented in this dissertation and embodied in 3Lisp can be considered to provide, at least in the first instance, only two kinds of referential access: *extensional* and *hyper-intensional*. The only entities that can be named or referred to, in 3Lisp, are: (i) entities that other structures or expressions name or refer to—the default case, in which one refers to them by using those other structures or expressions "transparently," in an extensional context; and (ii) other structures or expressions *themselves*, which is accomplished by using quotation (either «'» or «"…"») or the explicit NAME operator ('□'),[4] thereby obtaining both referential (φ) and causal (ψ) access to that entity as a causally-efficacious mechanical ingredient.[5]

Even though reference to intensions is not supported in 3Lisp, however,[6] that does not mean that intensional issues are off the

1. References are in the form *page/paragraph/line*; with ranges (of any type) indicated as *x:y*. For further details see the explanation on p.:
2. «Refer to the "three spellings of intentionality" sidebar—but where is that?
3. Included here as ch. 3c.
4. See 115/-2/1:2; regarding the use of "French quotes"—the characters '«' and '»'—cf. also footnote † on p. ·15 of ch. 4.
5. Except cf. A79, below.
6. That is: the ability, given any expression or structure x, to construct a differ-

table—as indeed they cannot be in any functioning computational system. As (rather verbosely) discussed in dissertation §4c.i (included here as ch. 3c) the foundational notion of λ-abstraction, and the behavior associated with the primitive closure bound to the name LAMBDA in the global environment, can only be understood in intensional terms.

As explained there, the "function" (i.e., something like *purpose*) of a composite expression[6.7] of the form '(LAMBDA *PATTERN BODY*)' is roughly to "capture" both the declarative and procedural intension that the expression *BODY* has in the the context where the compositive LAMBDA term occurs, in such a way as to allow that intension to be used or invoked in other contexts. The issue is that the intensions of expressions are in general context-dependent. Lacking a theory of intensions, I was not able to provide 2Lisp and 3Lisp with a mechanism with which to name, denote, or refer to the intension that *BODY* has in that original context.[6.8] What reduction of the expression (LAMBDA *PATTERN BODY*) in context C does, however, is to produce a structure S (a closure) that is co-intensional with *BODY* in C (i.e., has the same intension that *BODY* does in C), but that is intensionally (as well as extensionally) context-*independent*, so that S can be used in any other context C⍰ requiring a structure that has the intension that *BODY* had in C.

No matter how useful, however, a mechanism that allows one to construct context-independent co-intensional expressions is still a far cry from being able to designate such intensions explicitly. The latter was an explicit design goal for Mantiq, as described in ch. 2, for which 3Lisp was intended to be a design exercise. As mentioned there, moreover (cf. also annotation A3 of ch. 3a), one of the ways I imagined accomplishing this, in Mantiq, was by defining the struc-

. . .

ent expression or structure Y, such that the *extension* of Y is the *intension* of X.

6.7. In 2/3Lisp terminology, each occurrence of the term 'expression' in this and the next paragraph should be 'impression.' but I have used 'expression' for simplicity: as regards the issues at hand, the difference between expressions and impressions is immaterial. (The expression/impression distinction is a perfect example of the sort of distinction that the fan-calculus is designed to make only perspectivally visible.)

6.8. The reflective mechanisms did allow programs to designate the *context itself*, taken to consist of a processor continuation, an environment, and a structure being processed, plus the always available background structural

tural field sufficiently abstractly so as to be able to *fuse* structural identity with an otherwise-motivated notion of intension or meaning (so that testing whether two structures were structurally the same could stand in for—could serve as the subpersonal correlate for[7]—determining whether they meant the same thing). By defining the structural field sufficiently abstractly, that is, en route to what I conceived as a "field-theoretic" view of computation, I hoped at least to reduce, and perhaps ultimately even dismantle, the distinction between intensional and hyper-intensional reference.

Even at the time, I viewed traditional model-theoretic accounts of meanings in terms of functions from possible worlds to truth-values as too coarse-grained for semantical as well as structural reasons—insufficiently articulated as regards issue of deixis and context-dependent reference, etc.,[7.5] as well as raising computationally intractable issues of identity, and therefore inappropriate candidates for fusion.

I still believe that it would be well-worth exploring this design idea of adopting a "field-theoretic" approach to the structural field of a computational process that honours intensionally-motivated constraints—one that, at least if implemented on currently recognizable hardware, would need to be backed by sophisticated relaxation algorithms to compute these more motivated but less fine-grained notions of structural identity. The situation is hugely complicated, however, by the fact that I no longer believe that "*the* intension" of an expression or structure is a well-defined notion. At the very least, the declarative "route" (φ) from sign to signified needs to be considered as much more continuous, or at least as a path with arbitrarily many distinct "points" along it, in order to deal appropriately with the complex sorts of deictic and indexical dependence that natural language illustrates and that a system like Mantiq would require. An adequate exploration of these issues therefore remains for future work.[7.7]

A2 ·1/-1 While the first two issues (formulating an epistemologically adequate descriptive language and unifying that with a theory of computation or procedural consequence) were not addressed in 3Lisp,

field.

7. See A24, below.

7.5. Cf. ch.2, §...

7.7. Cf. the discussion of the fan calculus in the Introduction (§...) and in ch.2

as described in ch. 2 it was my intent to address both of them in Mantiq.

A3 ·1//-2:-1 For more discussion of the dual-calculus nature of Prolog see ·50/1:21/1.

A4 ·2/1/-3:-2 At this point, in describing "a program able to reason about and affect its own interpretation," I had not yet distinguished the various meanings of the term 'interpretation' that will be discriminated later in the chapter (in particular, descriptive vs. procedural meanings). In this context what I primarily had in mind was the more computationally prominent procedural notion of interpretation as *program execution*, rather than the representational or declarative sense familiar from logic and philosophical semantics.

A5 ·3/1/-4 As mentioned in annotation A2 (at ·.....) of the dissertation preliminaries (ch. 3a), to minimise confusion I explicitly flag chapter references that refer to chapters in the dissertation, of which only chapter 1 is included in this Volume,[8] so as to distinguish them from references to chapters in the present volume.

A6 ·3/-1/-3 As remarked in ch. 2, the writing in the dissertation is somewhat confused as to whether the declarative or representational interpretation of computational states had to be *externally attributed*—by, as it is said, we "external observers"—or whether that was only a contingent fact about current systems, with the possibility remaining open of computer systems achieving their own genuine or authentic *original intentionality*.[9] This and a number of other passages (e.g., see ·35/1/9:10) are written as if such semantics *had* to be attributed—an empirically reasonable enough point of view in 1981 (when the dissertation was written), given the state of existing computer systems, and the philosophical strength of the "formal symbol manipulation" construal of computation. But even though I did not feel that most practicioners in artificial intelligence grasped the gravity of what original intentionality would require (full-blown normativity, essentially), I was nevertheless already unquiet about the adequacy of the formal symbol manipulation thesis. As a result, therefore, as pointed out in other places (e.g., see ·44/1 and A24) it was not my intent to take as definite stand on the issue as these passages suggest. (Cf. also such passages as ·35/1/9, where I sug-

(§...), especially as regards "opening" and "closing" designation relations.

8. For reference to an internet-accessible version of the entire dissertation see p·... of the Introduction.

gest that some declarative interpretation was *tacitly* attributed—a separate and equally indefensible claim.)

A7 ·4/0/2:5 This comment that 2Lisp "makes explicit much of the understanding of Lisp that tacitly organises most programmers' understanding" would have been stronger if it had made the normativity more central—e.g., by saying that 2Lisp makes explicit how it is that its processing regimen *honours programmers' attribution of representational meaning* (just as, as described in §5c of the Introduction, in a sound inference regimen, the derivability relation honours the pre-theoretic attribution of referential meaning). That is: it would have been good to have make explicit the fact that programmer's understanding of Lisp is at least in part *deferential*. Cf. also ch. 2, §....

A8 ·4/1/4 See A3 (re ·39/n3/3:6) in "Reflection and Semantics in Lisp," chapter 4 «decide where to have original»

A9 ·5/1/9:10 Re "tacit attribution" see A6, above.

A10 ·5/1/-7 As noted in annotation A1 of ch. 3a, at ·.../..., it was misleading, and strictly incorrect, to suggest that declarative import and procedural consequence be *independently* formulated. My overall intent was for declarative import to precede procedural consequence, both ontologically and explanatorily (modulo issues about dynamicism, grounds for pragmatist epistemology, etc.; see the Introduction p. ·, and A·). The idea was that procedural consequence—what happens to program fragments, how they are executed—would then be defined so as to honour that declarative import, just as derivation is defined in a sound logic to honour semantic interpretation. Theoretically, the procedural consequence (execution) could be defined entirely arbitrarily, but in practice not only would that not be the intent, but it would vitiate both the whole point and the intelligibility of the resulting system.

A11 ·5/1/-5:-1 A number of reasons led my to call both dimensions *semantic*. Superficially, it was rhetorically important since, as mentioned at numerous points throughout this volume, computer science uses the term 'semantics' for the procedural dimension—based in part on its adoption of a specificational view of programs, with the consequence that an "interpreter" is taken in computer science to be an engine that effects the behavior that a program is taken to specify.

More substantially, though, in spite of giving representational semantics or declarative import both ontological and explanatory priority (cf. the preceding annotation), as mentioned in §3 of the Introduction, and examined in some depth in §... of ch. 2, I was nevertheless deeply respectful of the overall dynamicism about significance in general that computational experience pushes towards, and hence sympathetic to a pragmatist orientation towards both reasoning and ontology. As will emerge in §..., in characterizing 3Lisp I ultimately formulate a single overarching account of a computational process's significance, of which the declarative and procedural end up being aspects or projections, reciprocally tied together both structurally (causally) and normatively.

A12 ·5/2/7:8 By "recursively-specifiable but not compositional" I was referring to what is discussed in §6 of the Introduction. By "strictly compositional" I intended what I discuss there as the originating idea behind compositionality: that that is assigned as the "meaning" of an ingredient structure remains true to what, in at least some intuitive sense, the term, in a contextually-dependent way, means or refers to in that context, rather than to a full account of the contextual-dependence of that meaning or reference (the latter being what the "recursively-specifiable" algorithm needs to work with).

A13 ·6/1/9 In using the predicate 'abstract' to characterize internal or ingredietnal structures (the second analytic axis along which computational calculi are described), I did not mean to suggest that such structures are Platonic or otherwise immaterially diaphanous, but merely that the individuation of such structural elements can be, and in the 2/3Lisp case definitely is, "more abstract," in the sense of making fewer distinctions, than are neeeded to make sense of (the concrete materialities of) written textual or notation expressions. However the term "internal structure" (rather than "abstract structure" would have been more consonant with the ensuing discussion—e.g., where I say "each structural class be treated in a uniform way by the primitive processor."

Re the merits of and motivations for category alignment, cf. ch.2, §....

A14 ·6/-1:7/0 Although the dissertation asserts, in numerous places, that category alignment is an important aesthetic underlying the design of

2/3Lisp, the reason for that importance is not well explicated. There is a little discussion here, mostly about how failures to be category aligned tend to lead systems to resort to metasyntactic and metastructural access; see also the more extensive discussion at the end of [dissertation] §1.f.i, included here on pp. ·111:·112, as well as the discussion in §... of [this volume's] ch. 2. Nevertheless, after the first reports on 3Lisp were published, I was struck by the fact that the idea of category alignment was treated by most people I spoke with as at least extraneous, perhaps a distracting red herring, and perhaps even theoretically inelegant.

As usual, I believe the five things: (i) that the critical response has some merit; but (ii) unfortunately, perhaps because of that merit, the original proposal was too quickly dismissed or ignored, at the cost of understanding either of the underlying issue or of what the proposed solution was aiming to—and to some extent did—accomplish; (iii) that the concerns raised in the criticism are conceptually orthogonal to the merits of what was being proposed; and (iv) that as a result it should be—indeed is—possible to develop a solution that does justice to both the proposal and the critique; but (v) that, perhaps surprisingly, the development of a solution that simultaneously honours the original insight but avoids the issues raised in the critique will be much harder than one might initially suspect, requiring radical revisions in our overarching metaphysical and ontological frameworks.

What is right about the critique, to pick up the first of these points, is that it is endemic to programming practice to define more complex or abstract data structures out of simpler ones. Tying instances of such structures to the type structure of their implementation code not only fails to matter much, but contravenes some of the most important mandates of structured programming—that one not make one's code excessively implementation-dependent. Clearly, this aesthetic is deeply recognized in class-and object-oriented languages, and is embodied in the notion of an *abstract* data type. Whatever it is that motivates the category alignment mandate, therefore, should clearly be framed in terms of *user-defined classes or types*, not in terms of the primitive data structures that implement them. This is a very serious issue, discussed at some length in

§:, at :/.../. The class hierarchies paradigmatic of object-oriented languages provide some guidance towards how it might be approached; but because of their lack of declarative semantics what pursuing this direction would come to is not yet clear.

What is nevertheless right about category alignment, to turn to the second point—or at least right about the intuition that led me to propose it—are two things. The first, adduced in this paragraph (:36/-1:7/0), is that categorical confusion tends to lead to gratuitous use of semantic ascent, in the form of quotation and other practices fundamental to genuine reflection. This is the thrust of pp. :111:112, cited above; they point out how, in Lisp's case, a failure of category alignment leads to excessive (formally necessary but conceptually unwarranted) uses of quotation and calls to the primitive function APPLY. As these examples suggest, even at this relatively superficial level, sans category alignment, reflection gets very confusing, very fast.[10]

The second consideration motivating category alignment cuts deeper. It is almost fundamental to reflective processes that they tend to deal with the structures over which they are operating—i.e., the structures at one level below, which their own variables and data structures and arguments etc. denote—in terms of those structures' procedural and declarative semantical categories. Sometimes that is not true; sometimes there is something *particular* about an object-level structure or situation that requires dedicated focused reflective attention. But it is in the nature of things that it is typically in terms of the (more or less reified[11]) categorical structure of the object level domain that reflective procedures are most commonly defined.

Because of the formality condition, however—or anyway whatever it is that is right about the formality condition, whatever the formality condition turns into on a participatory construal, etc.—all that the reflective process has effective access to is structure, not semantics. And so being able to define *structural* type predicates that

9. Cf. the discussion at :33/3.

10. Cf. my comment «where?», in discussing the differences between 2/3Lisp and Scheme, that, en route to reflection, quotation needs first to be understood, then to be disciplined, and finally to be unleashed.

11. That is one of reflection's advantages, that it can reify what is implicit one level below. See A90, below, and §... of ch. 2.

sort structures according to their *semantical* character is critically important.[12] In the data-structure-oriented Lisp environment category or type alignment was a simple way of doing this. Once support for data abstraction is explicitly introduced, as for example it is in class systems and most object-oriented languages, more complex versions of such "alignment" would be simpler to provide.[12.5]

As mentioned in ch. 2,[12.7] the issue is that declarative semantics (φ) does not cross implementation boundaries—*including the "implementation boundary" separating an abstract data type or class from the structures and operations that implement it*. That implies that, in a system of such sort, when a programmer defined a category or class, they would not only need, as at present, to define its *behavior*, but would also have to specify its *declarative import* or *representational semantics*, or at least say enough about it to ensure that the programmer-specified behavior *honoured* (deferred to) *the appropriate norms*.[13] And to do that would require adversion to a fully adequate ontological theory of the subject or task domain of the program itself, rather than merely the subject or task domain of the language processor. I.e., it would require, or involve, or however one wants to put it, developing an account of *program semantics*, not merely of *programming language semantics*.

There is no doubt, as again discussed in ch. 2, that is a task that should be taken up. That was perhaps the main goal for 4Lisp, the envisaged next step in the series of design studies en route to Mantiq. But as stated briefly in chapter 1, the ontological problems proved daunting. The fact that I was not easily able to surmount them is why 4Lisp was never developed, and why Mantiq has yet to be designed. It was in part to address them that I wrote *On the Origin of Objects*, which from this perspective can be viewed as an attempt to discern a metaphysical framework that could be serviceable for a task of this sort. Developing the insights articulated there into a theoretical system that could form the basis of computa-

12. I once thought of this property as that of being *syncategorematic*, on the mistaken view that 'syncategorematic' mean *syntactic* (i.e., structural) *in virtue of categories*—or perhaps more simply, *syntactically (or structurally) categorical*. I still somewhat rue the fact that that was an egregious back-formation.

12.5. Simpler, perhaps—but not simple. Cf. «...».

12.7. See especially §... at :.....

tional analysis, design and construction is the task towards which the design of the fan-calculus[14] is oriented. *(cont'd)*

My belief remains strong that the resulting system would be powerful, useful, and elegant—as does my resolve to design it. Thirty years on from designing 3Lisp, however, I am not sure whether I can honestly yet say that I am more than about halfway there.

A15 ·7/1/8 Re 'independent': cf. A10, above, on passage ·35/1/-7, and annotation A1 of ch. 3a, at ·…/….

A16 ·7/2/-5:-1 It would have been simple, and for some readers helpful, to frame some of these points in terms of the philosophical concepts of *semantic ascent* and *descent*. Thus in characterizing the 2Lisp processor as semantically flat I am saying that, in 2Lisp and 3Lisp, normalisation (the default processing regimen) does not, but reflection does, engage in semantic ascent and descent.

A17 ·8/0/-4:-3 The distinction between an object language and a meta-language is invariably context-relative. Languages, including small fragments and/or individual expressions, do not intrinsically have the status of being at the object or meta level; they acquire any such status only in relation to another language or expression or intentional system. In simple contexts the meaning may be clear, but complexities invariably arise in any context in which reflection, implementation, theoretical description, and semantic ascent (and descent) are all simultaneously at issue. In 3Lisp, for example, code at each level in the tower is simultaneously object language from the point of view of the level above, and meta-language from the perspective of the level below.

In the passage in the text, however, the distinction being made is not between levels, but beween whole systems. In particular, by 'meta-language' I am referring to the external descriptive language that a theorist might use to describe or theorize 2Lisp or 3Lisp—paradigmatically, the mathematical λ-calculus. By 'object language' I mostly meant 2Lisp or 3Lisp—though in the case at issue, regarding the designation of environments and continuations, the object language would be specific passages of 2Lisp or 3Lisp that were nevertheless "meta" to some other code or processing that those passages were engendering or representing (at a reflective level, in code for a meta-circular processor, etc.).

A18 ·8/0/-2 Note that it is *designators* of environments and continuations that are part of the protocol code. There is a strong sense, because of the existence of these designators, that environments and continuations are *themselves* part of the operational definition of 3Lisp, but that is not strictly correct.[14.5] Better would be to say something along the following lines: that environments and continuations are extraordinarily good (realistically: indispensible) theoretical entities in terms of which 3Lisp's mechanical structures and ensuing behaviours can be found intelligible. However the truth of that statement should be not taken as implying that environment *structures* and continuation *structures* are a primitive part of 3Lisp. To speak in that way would be not only to confuse implementation with implemented (computer science's analogue of a use/mention error), but also to fail to appreciate the importance of the declarative dimension of 3Lisp (and 2Lisp) semantics.

A19 ·8/1/-2:-1 This characterization in terms of a limiting ideal is what in §1·e·iv I call a 'tower' view of 3Lisp; see especially ·101/1.

A20 ·10/0/-3:-1 I firmly believe the final statement in this paragraph: that nothing stands in the way of procedurally reflective, semantically rationalized versions of languages that support data abstraction, user-defined classes, and message passing. In spite of the promise made in the middle of the paragraph, however, the submitted dissertation did not show in detail how this could be done. As discussed in ch. 2, and at length in A14 (re ·36/-1:7/0), above, the theoretical challenges are considerable.

A21 ·11/2/-1 Dismantling the distinction between declarative and procedural calculi (the second design goal identified on the opening page of the chatper) is of course one of the goal of Mantiq—in effect the subject matter of this entire paragraph.

A22 ·11/-1/-2 Cf. A11 (·35/1/-5:-1) and the discussion in ch. 2 about the use of the term 'semantic' to characterize the unification of these aspects.

A23 ·13/-1/8 «Reference Dennett, Haugeland, Searle, as appropriate...»

13. Cf. the discussion of Amala in ch.2, §..., as well as §... in the same chapter.

14. Cf. §· of ch. 2.

14.5. I say "*operational* definition" because environments and continuations are not required to be mechanical or causal parts of the implementation. Overall, since I take computation to be intentional, and intentional systems to be constitutively (even if not intrinsically—cf. the discussion at «...») semantic, 3Lisp, in my book, is not merely its mechanical projection, but is constituted

A24 ·14/1/-3:-1 As stated in A6, above (at ·33/-1/3), the writing is less than clear about whether the declarative semantics *must* be attributed, or could at least potentially be original.

More important here, however, is the following: because reflection is defined in terms of φ, the question of whether or not a system is or is not a reflective system depends on what one takes to be the status of the declarative semantics. It is not a peculiarity of the 3Lisp approach to provide reflection in an architecture defined in terms of a double (ψ/φ) semantical account, in other words. Rather, the notion of reflection—of a system reasoning or engaging in process that is *about* its own operations and structures—requires a prior, non-procedural notion of *aboutness*. Cf. the last sentence in the subsequent paragraph, which talks of systems dealing with their own ingredient structures and operations *as explicit subject matters*.

Hence my sense that, no matter how otherwise gracious, Friedman entirely misses the point in attempting to define a notion of "reflection without the metaphysics" «ref» (though cf. also A...)

A25 ·14/-1:46/0 These two paragraphs are wordy and confusing. See the next annotation (A26).

A26 ·16/0 It would helped, rhetorically, if these two long paragraphs (·44/-1:46/0) had been phrased in terms of the personal/subpersonal distinction (a framing I of which I was unaware at the time).[15] Even then, though, the issues are subtle.

If, *qua* person—i.e., at the personal level—I think about Virginia Falls, then according to the Knowledge Representation Hypothesis (KRH) that happens in virtue of my brain's constituting of a subpersonal process P (formally) manipulating equally subpersonal representations of Virginia Falls. My personal-level *thoughts* about Virginia Falls, therefore, have, as their subpersonal correlates, something like *subpersonal symbolic representations of Virginia Falls*—i.e, subpersonal interior symbols denoting the falls. If therefore, at the personal level, I then *reflectively think about my (personal) thoughts about Virginia Falls*, then by the KRH that must happen in virtue of, at the subpersonal level, an internal processor P *manipulating internal symbols representing those personal-level thoughts*.

in part by its semantic interpretation. And there is no doubt that environ-

Now at this point the 3Lisp architecture makes an assumption, which I will call γ, that it is important to spell out. Depending on one's viewpoint, one might either characterize γ as so obvious as barely to deserve mention, or indict it as a devious sleight of hand. Call my personal-level thoughts about Virginia Falls T_1, and their subpersonal correlates R_1. Similarly, call my reflective thoughts about my thoughts about Virginia Falls T_2, and their subpersonal correlates R_2. What assumption γ has to do with is the declarative semantics of R_2.

According to the KRH, R_2 should be an internal representation of T_1. Instead, what the 3Lisp architecture presumes is that we can treat R_2 as—can assume it is equivalent to, can in some sense take it to *be*—a representation of R_1. That is, we can define γ as follows:

γ Instead of taking the subpersonal correlate of a reflective thought to represent a personal object-level[16] thought, we instead take it to represent the *subpersonal correlate* of that object-level thought.

Assumption γ underwrites the semantic interpretations of reflective structures presented in the rest of the dissertation. By stipulation: (i) $\varphi(T_1)$=Virginia Falls themselves, the cascading sheets of water; and (ii) $\varphi(T_2) = T_1$, my falls-directed personal-level thoughts. According to the KRH, $\varphi(R_1)$ is the same as $\varphi(T_1)$—that is, the same cascading sheets of water.[17] T_1 and R_1 are distinguished by *type* or *form*, that is, not by *content.* The former are thoughts, the latter are (formal) representations, and both designate the same thing—namely, waterfalls. At the meta or reflective level, however, the preconditions arise for conceit γ. By the KRH, $\varphi(T_2)$ and $\varphi(R_2)$ should again coincide; they should both be T_1. By γ, however—and thus what is embodied in the 3Lisp architecture—instead of having $\varphi(R_2)=T_1$, we have $\varphi(R_2)=R_1$.

This way of looking at things suggests that γ is a cheat. But there is another way to understand γ, according to which γ is not only far less problematic, but actually well-motivated. Instead of taking subpersonal meta-level representations to represent other subpersonal symbols (instead of representing the personal-level thoughts those

ments and continuations are part of the 3Lisp dialect, so conceived.

15. «Reference: Dennett—was he first?»

16. Cf. A15, above.

symbols are correlated with), one could instead say that really, as required be the KRH, they *do* represent the personal level thoughts, but do so in virtue of bearing a closely-allied but nevertheless distinct relationship, pretty much like designation, to the corresponding subpersonal correlates. Label that "closely-allied but nevertheless distinct relationship" φ'. Then, using the example above: if we take $\varphi(R_2)$ to be T_2, we would have the following:

1. $\varphi(T_1)$ = Virginia Falls — stipulation
2. $\varphi(R_1)$ = Virginia Falls — (1) plus KRH
3. $\varphi(T_2) = T_1$ — stipulation
4. $\varphi(R_2) = T_1$ — (3) plus KRH
5. $\varphi'(R_2) = R_1$ — proposal

That is, as these equations make clear, φ' would in a sense be the *subpersonal correlate of designation* (φ).

What is haunting about this admittedly arcane discussion is that in teasing apart distinctions (e.g., between personal-level thoughts and their subpersonal correlates), and then following up the logical implications of so doing, we end up teasing apart other things (designation and its subpersonal correlate), in a process that is reminiscent of the very problems that 3Lisp inadvertently introduced, by being a stickler for use/mention distinctions. That is: whereas 3Lisp was rigorous about distinguishing signs from what they signify, to the point of ultimately becoming unusably fastidious, the present discussion is doing the same thing as regards the personal/subpersonal distinction.

I believe the morals are profound. As argued in ch. 10, computational systems are permeated with cascades of relations between and among entities or relations that for some purposes can be seen as sufficiently similar so as not to warrant making a distinction between them, and for other (perhaps rare) purposes distinguishable (e.g., φ and φ', in the case at hand). To put it in the terminology of *On the Origin of Objects*,[18] for some purposes it may be important to register a distinction between a thought and its subpersonal correlate; for other purposes, not. The challenge is to frame the background metaphysical/ontological/epistemological assumptions, and the working theoretical framework, in ways

17. That is what it is to say that personal-level thinking is constituted by a subpersonal processor manipulating internal *representations*.

that can honour this approach of doing justice to distinction in a (normatively-driven) context-dependent way. The former, of course, is the aim of O3; the latter, of the fan calculus project suggested in the Introduction.

A27 ·16/1 This paragraph, too,[19] would have been more clearly explicated in terms of a personal/subpersonal distinction. At issue is what subpersonal activity underwrites or is correlated with a personal-level reflection. On the surface, the claim is this: it does not consist of interior process P *reflecting* on R_1, which would seem circular. Rather, the architectural idea is that P considers R_2.

What is striking is that this claim is framed with reference to the level-shifting view.[20] The story on the tower view is more interesting. In particular, on the tower view, one could say that, sure enough, when, at the personal level, someone has a reflective thought T_2 about base-level thought T_1, that happens, subpersonally, in virtue of the person's interior (subpersonal) process P_1 *reflecting* on subpersonal correlate R_1. But then we discharge the threat of circularity by offering an account of what it is for subpersonal process P_1 to reflect on R_1. Specifically, we *iteratively apply the KRH*, and say that P_1's reflecting is "sub-subpersonally" realized in virtue of P_2 manipulating R_2.

While it has real merit, positing such an iterative application of the KRH is also somewhat ironic. It brings up the point made «where? at least point back to §6 of the Intro. also ·105/1?» that one virtue of substantial architectures is that some questions need not be answered. But it ties in as well to the point made in the previous annotation; that the very distinction between the level-shifting and tower views is ultimately a choice between two behaviorally-equivalent registrations.

Perhaps the most important moral, in this case, is that the rough equivalence of these two views begins to undermine the integrity or anyway absoluteness of the personal/subpersonal distinction itself—a point towards which many other issues point as well.

A28 ·16/-1/-3:-1 Cf. ch. 6.

«Needs work. Cf. comments in the POPL paper. Also see earlier annotation, and comments at the beginning, about the relationship

18. See also "Representation and Registration," ch.… of Volume ii.

19. Cf. the former annotation, A26.

between self-reference, introspection, and reflection...»

A29 ·17/2/8 Snobol ("String Oriented Symbolic Language"), a string-processing language developed at AT&T Bell Laboratories in the 1960s, had the distinctive property of allowing strings to be treated as programs, thereby enabling programs to be dynamically constructed and executed on the fly. Famous for treating patterns as a first-class data type, Snobol served in some ways as a precursor to such modern scripting and text-oriented languages as Perl.

A30 ·19/0/-1 The term 'formal' in this sentence means something like "rigorously technical" or "mathematical" or something of the sort—not syntactic. Cf. the discussions of diverse meanings of 'formal' at ·.....·.

A31 ·22/-2/-4 This is just one of several places throughout the chapter[21] where I talk about reflection needing "actually to matter" to the process in which it occurs. Unfortunately, the sense that I had in mind, or at least that I could make good on at the time, was more one of *having concrete physical consequence* than of *being important*. I was perfectly well aware that a more encompassing normative sense of mattering was fundamental to the long-term goal of genuine reflection.[22] However, as discussed at numerous places in this chapter (e.g., see ·43/-1), I was under no illusion that 3Lisp achieved the requisite semantic originality that genuine mattering would require. For these reasons, it would have been better to have written more modestly of something like *effective consequence*. (See for example the discussion in §1b.iii.ζ, at ·29/1).

A32 ·24/1/9:10 The ability to plant seeds, now, so as to ensure future reflection is tremendously important—not only to the design of 3Lisp and other procedurally reflective systems, but for an understanding of reflection in the general case. It is certainly a staple of mundane personal psychology: to be able, in advance, to "set oneself up" so as to ensure that later, when some circumstance arises, one will at that point stop and reflect (e.g., about what to do or not do).

A33 ·25/-1/-4 The claim that in ordinary Lisps environments and continuations are not "internal to the language" is incautiously stated. For one thing, what is clearly meant is that environments and continuations are not *objectified* "within" the language—do not exist *as discrete objects*. Whether they exist as relational or configural realities de-

20. For a discussion of the level-shifting and tower views, cf. §1·e·iv, especially ·101/1·.

21. See ·59/0/10·, ·62/0/4·, ·62/-1·, and ·104/1/-3:-1·. «...also refer to annota-

pends on how the language is registered—i.e., can only be answered with respect to a theory of the language, rather than being a fact of the language "per se" (as if such a thing existed, or made sense). In addition, by "within" or "internal" I clearly meant as part of its mechanical/behavioural footprint, rather than "complete with (declarative) semantic interpretation." The tacitness of this assumption would have confused no readers; but given my commitment to viewing computation as intentional I should have expressed the point more precisely. (Cf. annotation A..., above.)

A34 ·26/2/4:8 The claim that no computational process can achieve the limit of reflexive (as opposed to reflective) thought is something I thought at the time the dissertation was written (1981), and so I have left it standing. However, it is not a statement I would blithely endorse in 2014. Increasingly, I have come to believe that there *are* ways in which it is possible to have a "I am now thinking" thought refer to itself, or perhaps more accurately to *include itself within its referential domain*, without invoking a Necker-cube like reverberation between one state and another—perhaps in something like the way in which non-well-founded set theory[23] supports the notion of a set having itself as a member. Descartes' *cogito*[24] does not seem semantically ill-formed, after all, even it feels a little phenomenologically unstable—though the shifting-back-and-forth that accompanies thinking about it may derive from a double self-reference, involving not only (reflexively) thinking that one is thinking, but also (reflectively) thinking—or recognizing—that one is in fact doing that.

The epistemic achievement necessary to genuine reflexion, I believe, would involve entertaining a reflexive thought *quietly*, as it were—to hold a self-encompassing thought, in such a way that awareness of its reflexive self-referentiality (or semantic self-inclusion) does not lead one into a kind of vibrating or alternating epistemic state. While not necessarily easy, I believe not only that this can be accomplished, but also—perhaps oddly, perhaps not—that doing so relates to a number of forms of self-referential discipline that have been developed in various Asian and other meditative and mystical traditions. At a more mundane level, and apparently unlike some others, I do not believe that either the meaning or the truth of such statements as that "all statements are perspectival" need in

tions A36, A38, and A39»

22. See e.g. Haugeland's "Truth and Rule Following" «ref».

any way be undermined by the fact that they apply, among other things, to themselves.

As explained in more detail in ch. 6, I characterize as 'reflexive' not only those states, processes, expressions, etc., that are strictly *self-referential*, in the sense of being their own semantic extension (e.g., "this very five word phrase"), but also those that we might call *self-applicable*, in the sense of including themselves within their referential or semantic extension (such as "all phrases in English"). By 'reflective,' in contrast, as here, I refer to processes of "stepping back" and assaying, from a distinct vantage point (cf. A·…), another part or aspect or period of oneself. There is no doubt, according to this distinction, that even if computational models of reflexion are possible, 3Lisp was correctly described as a model of computational reflection. (Cf. also §1·b·iv, starting on p. ·59.)

A35 ·28/2 Cf. the previous annotation (A34, re ·56/2/4:8). As always, "stepping back" must be defined with reference to an encompassing frame of declarative or representational semantics (φ), and hence, because of semantical deference, depends on a background structure of norms. While it is true, as mentioned in the previous paragraph in the text, that one cannot step outside of, for example, language, that does not imply that one cannot achieve a normatively governed vantage point from which to regard language with some detachment. It is just that that detachment will not be complete (at which point I would call it *disconnection* rather than detachment).

A36 ·29/1/11 Cf. A31, above (·52/-2/-4).

A37 ·30/-1:31/0 Cf. the discussion of "broad" vs. "narrow" self-reference in §… of ch. 2 (·…/…).

A38 ·32/0/4:6 Cf. A31, above (·52/-2/-4), as well as A36. As is evident here, the reason for the internality of the causal relationship has more directly to do with effective procedural consequence than with anything authentically normative (though of course the former is required in order to honour the latter).

A39 ·32/-1:33/0 Cf. A31, above (·52/-2/-4).

A40 ·33/-2/7:8 At the time this was written, I was already starting to reject the claim that computational processes are *formal*, in the sense of operating independently of their semantic interpretation, but I had yet to question another widespread assumption: that computational ar-

rangements are *abstract*—or anyway, as is being said here, temporal but not otherwise physical. I have come to profoundly disagree with this view, believing that computational processes are as much denizens of the material or physical world as, for example, are we. See both 03 and AOS.

A41 ·34/1/7:8 «Reference the discussions in other papers—POPL? Prologue? CC? I forget where this is talked about, complete with those α/β figures, etc.»

A42 ·35/2 The "process reduction" model presented here might informally be taken to be *reductionist*, in the informal sense that it analyses something (processes) in terms of their ingredients (other processes, plus interactions and structural fields of symbols). Not only is it far from the notion of reductionism that has come in for attack in numerous discursive traditions, however; it does not even meet normal criteria on being naturalistic, since the ingredient processes, being themselves computational, are assumed to be semantic.

More substantive is the question of why I called these serial and parallel *reductions*, rather than serial and parallel *implementations*, which would be the more usual framing. I still think the 'reduction' phrasing has merit, in putting the emphasis on how the (exterior) process is *understood*, rather than focusing on the mechanical issue of constructing it.

A43 ·36/1/3:4 In the dissertation I called these *interpretive* and *communicative* reductions, respectively, but in retrospect that choice of terminology seems strikingly ill-advised. For one thing, to employ the adjective 'interpretive' for those processes that contain a single interior locus of agency is too committed to the computer science sense of the term 'interpreter,' which I have explicitly set aside in favour of 'processor.' Similarly, to employ 'communicative' for interacting multiple interior loci of agency commits to their interaction being one of "communication"—i.e., to involve the exchange of *meaningful* entities, which is a more specific claim about the nature of process-process interaction that I want to be committed to at this level.[25] The predicates "serial" and "parallel" seem both simpler and more consonant with general computational practice. Because this termi-

23. «Ref Axcel, others?»

24. Independent of the 'ergo sum' part.

25. By analogy, cf. John Haugeland's characterization of digitality («ref») in terms of 'reading' and 'writing'—a similarly unwarranted use of intentional

nological change reflects a substantial intervention, however (not merely a correction of spelling, e.g.), I have marked the uses of both terms with brackets here and through the rest of the chapter.

A44 ·36/1/6 In the dissertation this was written "how these processes *communicate*" (emphasis added); I have changed it to 'interact' in line with the previous annotation (A43).

A45 ·36/1/9 This is the lay English rather than technical computer science sense of 'interpreted'; I would now say "registered."

A46 ·37/1/-6 Cf. also ... «Point also to other papers and commentaries as appropriate»

A47 ·37/-1/4:6 I continue to believe that the relation between *programs* and programming *languages* is of far more theoretical importance than is normally recognized. See ch. 2 for a discussion of the relation between the semantics of programming language and the semantics of particular programs. «...»

A48 ·41/1/4:5 The last sentence in this ¶ is too strong. I had not yet developed the language of *registration* (cf. O3 and "Representation and Registration", ch.... of Volume II). In its terms, I would rephrase the first sentence of the paragraph as follows: "To implement Lisp, in other words, all that is required is the provision of a process that can be registered as consisting of the Lisp structural field and the interior Lisp processor." So stated, the claim would be so obvious as not to have been worth making. The point is that the dissertation was written in the grip of an untenably naïve realism, which occasionally (e.g., see A55, below) required caveats and maneuvering to sidestep.

A49 ·41/-1/3 The use of 'ψ' in this paragraph has nothing to do with the use of the same symbol for the procedural consequence portion of full semantical significance in the 2/3Lisp framework (i.e., of the ψ/φ pair). For this example it would have been better to use a different label.

A50 ·41/n11 The reference was to Fodor (1983).

A51 ·42/1/2:3 The characterizations of interpreters (in the computer science sense of the word), compilers, etc., given over the previous several pages, are all framed behaviorally, rather than in terms of declarative or representational semantics. And so it might seem as if a mathematical theorization would require formalising only ψ, not φ or a more generalised significance function Σ. But as discussions throughout

vocabulary, in my view, to characterize forms of interaction that are not nec-

the chapter make evident, I believe that no analysis that does not treat φ and declarative import generally would get at the heart of the computational phenomenon.

A52 ·42/2/2 I believe this statement is false. Locality must be defined, but only topologically; to require a metric or measure is too strong.

A53 ·42/3/7 As reported on Wikipedia,[26] the first evidence of a COME-FROM instruction appeared under the label 'CMFRM' in humorous lists of fraudulent assembly language instructions. It was elaborated upon in Clark, R. Lawrence, "We don't know where to GOTO if we don't know where we've COME FROM", *Datamation*, 1973, written in response to Edsger Dijkstra's "Go To Statement Considered Harmful" «ref».

A54 ·43/1/-10 The sense of interpretation intended on the line, and throughout the paragraph, is clearly *declarative semantical interpretation*—not the procedural processsing sense of computer science.

A55 ·43/1 This and the subsequent paragraph are clearly an informal and not especially clear amalgam of Fodor's *formality condition*, Dennett's *intentional stance*, and a distinction between *original* (*authentic)* and *attributed* (*derived*) intentionality (Haugeland ...). Fodor's classic formulation of the formality condition appeared in 1981, the year this dissertation was written;[27] Dennett's Intentional Stance was not published until six years later (Dennett 1987), though formulations had appeared earlier.[28]

For numerous reasons I no longer believe that 'computational' is best understood as a predicate on explanations. My current sense is that the best way to understand the reason I thought so then is that in 1981, as mentioned in annotation A48 above, I was still in the grip of an excessively naïve realism—or anyway did not yet have vocabulary in terms of which to say anything different. The language of registration (03) would have allowed the points to be much more simply and effectively presented.

Note, moreover, even setting aside issues of theoretical vocabulary, that the thesis that computational semantics is necessarily attributed or derivative (see A6, above) does not imply that 'computational' is a predicate on explanations. As I have said elsewhere,[29] to say that intentionality is derivative is not to say that it is not *real*;

essarily intentional at all.

26. «Ref»

27. «Fodor 1981.»

it is real *as derivative*. A theory of derived intentionality can still be perfectly ontological; it would just need to explain the ontological conditions *for the interpretation being attributed*. (In terms of metaphysical status, all that derivative intentionality denies is that the intentionality is *intrinsic*—but that is a different thing. It would be a stringently impoverished metaphysics, to say the least, that restricted reality to the intrinsic.)

The main point in the text, however, is that the fundamental thesis argued in the dissertation—that reflection is straightforward to understand and implement if built on a semantically clear base—implies that developing an account of computational reflection, and hence designing 3Lisp and like languages, requires not only understanding such philosophical views about the nature of computing, but effectively "building them in" to the resulting reflective architecture.

A56 ·43/-1/-3:-2 The term 'syntactically' is used here as in philosophy, not as in computer science. At best, in computer science one would say 'structurally,' but the meaning would be so deeply assumed that to dignify it with a label at all would seem odd.

A57 makes no sense!

As mentioned in several places (e.g., see ch. 1), I no longer believe that the formality condition is correct, but was still under its influence in 1981.

A57 ·45//-2:-1 As explained in ch. 2, at the time this dissertation was written, perhaps in part because the project emerged from several years working the area of knowledge representation, I was singularly focused on an *ingredient*ial view of programs. I did not considered the position, much more commonly held in computer science, of viewing a program as a *specification of*, rather than as an *ingredient within*, a computational process.

A58 ·46/-2/1:2 Philosophical readers will find it awkward to characterize both aspects as *semantic*. Cf. A11, above (re ·35/1/-5:-1).

A59 ·47/1 This paragraph—and in fact most of this section (1.d.i)—is an early (to say nothing of rather inarticulate and embarrassingly verbose) attempt to explain what in the Introduction I call semantic *deference*.

A60 ·47/2/-5:-4 «probably Situations and Attitudes, but maybe one of the earlier papers—check»

28. «Check, and reference if appropriate»
29. «Where?»

A61 ·47/-1/-7:-5 It is because 'CAR' is being used as a term of English in the text, rather than as a Lisp identifier, that it is formatted in this paragraph (three times), and at various points elsewhere throughout the chapter,[29.5] in a serif rather than sans-serif font—i.e., as 'CAR' rather than 'CAR.'

A62 ·48/n12/1 The referenced postscript is contained in 3.f.iv (p. 246–252) of the full dissertation—not reproduced here, but to which a link is provided on p. .... Its subject matter is that mentioned above at «...», and discussed in §... of ch. 2: how to extend the declarative aspect of semantics to deal with abstract data types. As well as investigating some examples (e.g., the use of a pair of real numbers to model complex numbers—or more accurately, the use of a pair of "real numerals" to "implement" a "complex numeral"), the main point of the section is that if the denotation (φ) of a complex data type is defined in terms of the denotations of the structures in terms of which it is implemented, then the normalisation of a term representing an instance of the abstract data type will (in 2Lisp and 3Lisp) still be designation-preserving.

The topic is complex, however. For example, the disavowal of the distinction between syntactic and semantic domains points towards issues considered under the topic of 'use' in ch. 2, §....

A63 ·49/0/1:3 Cf. §5c of the Introduction, on p. ....

A64 ·49/1/5 In §... of ch. 2 I criticize the distinction between *narrow* and *wide* psychology adverted to here, on the grounds that the former is usually thought to have to do with events and processes ocurring within the head, in contrast to the latter (pscyhology widely construed), normally considered to involve reference to the external world. The φ/ψ distinction being introduced here is not vulnerable to that critique, however, because no assumption is being made that ψ involves only computationally-internal operations, or that φ is restricted to relations to external events and phenomena. On the contrary, 3Lisp's entire reflective edifice is constitutively characterized in terms of computationally-internal referential (φ) relations.

A65 ·49/-1/-3:-1 The statement that I will consider cases where S_1=D_1, while strictly correct, is misleading. While it is true that self-reference of this sort is discussed from time to time, such as in §1·b·iv (pp. 59–

29.5. E.g., see ·98/-1/2, and ·117/-2/4, and ·188//-3 in ch. 3c, as well as ·189/0/1 and 5 for similar English (descriptive) uses of 'CDR.'

30. 'Fortunately' because the traditional type/token distinction is profoundly too simplistic to deal appropriately with the range of one-many relations than

63), the topic is raised mostly in order to contrast it with the sorts of self-reference with which the study of reflection is concerned.

A66 ·50/0/1 Strictly speaking, my reply "Mark Twain" is presumably an *instance* or *use* of the referent of 'the pseudonym of Samuel Clemens,' on the assumption that the latter refers to a *type*—but the point is clear enough. In general, as pointed out in the Cover («ref»), the dissertation (fortunately[30]) pays little attention to type-token distinctions.

A67 ·50/0/5 See chapter 8 of *On the Origin of Objects*, pp. 243-67, for an extended discussion of designation-preservation under the label "preservation of reference."

A68 ·50/0/-5:-4 To claim that derivability (⍰) is designation-preserving is sloppy phrasing. What I meant was that, in the ordinary course of things, derivability is not a "level-crossing" operation. One can interpret the claim in a more mundane way as saying that if, from $\alpha_1 \ldots \alpha_k$ one were to derive β on the grounds that β is true if $\alpha_1 \ldots \alpha_k$ are true, then derivability has preserved the designation *truth*. But as well as being vapid, this is false; from falsehood one can derive anything, including claims that are true. But that was far from my intent.

A69 ·50/1/-7 "Crucially distinct" but of course normatively related; cf. §5c of the Introduction.

A70 ·53/-1/5 There is only one numeral per number within any given 2Lisp structural field; there is no need for multiple tokens or instances. So if, within the context of a given process P_1, there two distinct composite structures—such as for example '(+ 7 3)' and '(if (= x y) 7 9)'—the occurrences of '7' are structurally identical, rather than each having its own distinct "token" or "instance" of a common type. Think of the complex structures as manifolds that encapsulate the self-same unique numeral. That identity does not make the numeral into a type, of course; a different 2Lisp process P_2, with a distinct structural field, would in some sense have its "own" numeral 7.

The more flexible forms of identity possible within structural fields makes the issue of types/tokens/instances/uses of types—that is, the question of what is *one* and what is *many*—spectacularly more complex in computational cases than in the familiar case of written, lexical expressions. Theorizing such complexity, and providing facilities for viewing it as contextual or perspectival, are primary aims of the proposed fan calculus.

A71 ·54/1/-8 I put "operation" in quotes because of the claim that, *au fond*, quotation should be understood as a referential or naming convention, not as a procedure. As discussed in «where?», the operational consequence (ψ) of quotation should be defined, derivatively, terms of its referential function, rather than—as is so common in programming languages—being taken as a primitive behavioral operation.

A72 ·55/0/4:5 The λ-calculus does not provide for the definition of names. Functions can be designated only by use of the complex expressions that are needed to designate them. What I had in mind, when writing that we could "remove atomic designators," was that we could define a variant of Lisp that, like the λ-calculus, did not allow definitions, but instead required use of such full composite expressions in every function position. But the phrasing in the text—"the ability to *name* composite expressions as unities" (emphasis added)—is doubly confusing, and wrong. It is not that in a dialect that allows definitions one names *composite expressions*; rather, one introduces unitary names to name what those composite expressions denote (i.e., functions). Secondly, the point is not that one names things as unities, but that names are ways to refer to (arbitrary) things *with* unities. What is unitary is the name, not the named. What I should have said is that, in the λ-calculus, one *denotes functions with composite expressions instead of with (instances) of unitary names*.

A73 ·56/1/1:2 The story would end only if one could then show that the procedural consequence (ψ) honoured the declarative import in a normatively appropriate way; see §5c of the Introduction.

A74 ·56/1/-7:-6 Cf. the discussion of intensionality (i.e., 'intensionality-with-an-s') in A1 (·31/-1/5), and in dissertation section 4·c·i, includeded here as ch. 3c.

A75 ·56/2/6:8 As is true in so many aspects of the design of 2Lisp and 3Lisp, there may seem to be a considerable discrepancy between the underlying philosophical motivation for this point (about procedural consequence affecting the context of declarative interpretation) and the almost triviality of the technical examples brought forward as illustration. In making this particular point, and more generally in reciprocally defining declarative import (φ) and procedural consequence (ψ) within an overarching general significance function (Σ), as described in the next paragraph, I meant to do justice to

the dynamic, contextual dependences of reasoning and language in general—such as our needing, in rational thought, to keep up, in a fully participatory way, with a constantly evolving world, some of which changes are the result of our own doing. Examples would range from such simple examples as mundane temporal dependency (using 'yesterday,' tomorrow, to refer to what we today refer to with 'today') and performatives ("I promise to bring you coffee") to the sorts of consideration that underlie pragmatist epistemology in general. Causing a side-effect to a variable hardly connotes the richness of the phenomenon.

A76 ·58/-1/1:3 This is the equivalent, in a computational context, of saying something that would be obvious, logically: that one cannot specify a sound proof procedure (⍰) without first having in mind an interpretation function for it to honour.

A77 ·58/-1/3:7 This is too strongly stated. Full independence is not required; the two accounts could be reciprocally co-constituted. What I should have said is that defining a processing regimen in a calculus in which there is *nothing more* to meaning than "how the symbol or structure is treated" would not just evacuate the system of any semantic or intentional interest; it would also deprive it, in my view, of any claim to being a computational system at all—instead, reducing it to "naught but mere mechanism." Oil refineries, after all, are constructed of parts that have procedural consequence.[31] Computation, in my book, in spite of its abiding concern with mechanism and effectiveness, is nevertheless a fundamentally intentional phenomenon. (Cf. §5c of the Introduction, and AOS).

A78 ·63/0/-3:-2 For clarity, this should have been written as "reserving the word 'interpret' for the declarative interpretation function φ."

A79 ·69//1 An important property of reflective procedures implicit in this model was not adequately explained in the dissertation.

As will become increasingly evident as this chapter proceeds, and is spelled out in detail in the dissertation's remaining chapters,

permeate computational systems—one of the primary motivations for the development of a fan calculus.

31. Someone might object that programs specify or represent the behavior they result in, whereas mechanical parts, of the sort out of which oil refineries are built, simply have causal consequence. Perhaps that is so. The point is that, in order to make out a distinction like this between *specifying behavior* and merely *leading to behavior,* one needs an account of what it is to represent or specify (i.e., something like φ).

reflective procedures are used in 3Lisp in those cases that would involve the use of *intensional* procedures in non-reflective languages[32]—i.e., procedures that, in computational (ψ) terms, "do not evaluate their arguments," or, to put it more philosophically, procedures whose argument positions are opaque or intensional, rather than transparent or extensional. Quotation is a paradigmatic intensional operator, but others are ubiquitous, such as: desires, belief reports and other statements of epistemic state (such as memory reports) in natural language and thought; possibility and necessity operators, in logic; and LAMBDA, classical IF statements, "left-hand-side" expressions in assignment statements, etc., in computing. In 3Lisp, reflective procedures are used to subsume all such intensional practices.

One might expect it to follow that 3Lisp reflective procedures would not "evaluate" (i.e., normalise) their arguments—or again, to put it in philosophical terms, that the argument positions of reflective procedures would be opaque or intensional. Interestingly, that is not so—at least not in any simple sense.

Note that line 18 of the RPP is the only place where reflective procedures are ever invoked (that is: where their arguments are bound, their closures expanded, etc.). And as figure 15 makes clear, in that context, they are invoked perfectly *extensionally*—their argument positions are perfectly "transparent." In fact from a certain perspective one can correctly say, of 3Lisp, that *all* procedure calls are extensional—that, ultimately, there are *no* opaque or intensional argument contexts at all.

What is going on is this. If, in the course of regular "user" or "object-level" code,[33] the processor encounters a redex of the form (PROC α_1 α_2 ... α_k), where PROC names a "reflective procedure"[34] or is bound to a reflective closure, then, before so much as glancing at the expressions in argument positions α_1 α_2 ... α_k, the processor effects a level-shift—"backing up," to use the language of the Prologue—so as to obtain an appropriately detached vantage point from which to consider the situation. In 3Lisp, there is exactly one such "appropriately detached vantage point": line 18 of the RPP. From that vantage point, the structures in argument positions α_1

32. Including not just 2Lisp and all prior dialects of Lisp, but all programming languages other than 3Lisp.

$\alpha_2 \ldots \alpha_k$ are then *referred to*, perfectly extensionally. Sure enough, to switch again to philosophical jargon, those structures are *mentioned*, not used. But—and this is the important point—*mention* is a perfectly valid form *of genuine extensional reference*. It is genuine extensional reference to the expressions or structures that are occupying argument positions $\alpha_1\ \alpha_2 \ldots \alpha_k$.

More generally, that is, the 3Lisp architecture illustrates a non-standard approach towards opaque or intensional contexts. Traditionally, we assume that functions or operators that take their arguments in an opaque or intensional context work in the following way: (i) they treat their arguments differently from standard-issue (extensional) functions or operators, but (ii) they do so within the context of the interpretation of the sentences or complexes in which they occur. In 3Lisp, in contrast, a "reflective redex" in the sense just described[35] is understood as follows: (i) its occurrence signals a shift in interpretive context, to the reflective (meta) level, but (ii) in that different context the function or operator treats its arguments in the standard transparent way. In sum, rather than viewing opacity as a *different kind of reference* (reference is always extensional, in 3Lisp—that is how reference is taken to work), 3Lisp views it as a *change in interpretive context*.

I have not explored the merits or consequences of adopting this alternative view of opacity in logic and language more generally—but I believe it is a project that would be worthwhile. It is therefore included on a list I maintain, to use a little Pennsylvania Dutch, of "PhD dissertations needing written."[36]

A80 ·71/1· This paragraph is the one place in the chapter where I have added entire sentences to the text, in order to make the intended point clear. In the original dissertation, this paragraph consisted, in its entirety, of:

> "We will not take a principled view on which account—a single locus of agency stepping between levels, or an infinite hierarchy of simultaneous processors—is correct: they turn out,

33. Cf. a15 (re ·38/0/-3:-2).
34. The reason for the quotation marks will be explained in a moment.
35. I.e., a redex of the form (PROC $\alpha_1\ \alpha_2 \ldots \alpha_k$), where PROC names a reflective procedure or (equivalently) is bound to a reflective closure.
36. The example illustrates a general point made in the Introduction: of how

rather curiously, to be behaviorally equivalent. For certain purposes one is simpler, for others the other."

The terminology of 'level-shifting' of 'tower' views is discussed in §1·e·iv (see especially ·101/1), and also in chs. 4 and 5.

…… put in a discussion of how important this is, and not adequately emphasized in the dissertation: that the tower is the normative ideal, that supplied the "specification" that the implementation has to meet. It is another case of the normative (deferential) orientation that underlies all of this work…

A81 ·75/4 This entire paragraph presages concerns about perspectival object individuation discussed in AOS and targeted by the fan calculus.

A82 ·76/1/-2 In the original dissertation, the following parenthetical comment was inserted at this point (following the words "and so forth"): "It is important to recognize that the suggestion of constructing a reflective variant of the λ-calculus represents a category error." A few years later, however, contrary to this statement, I did informally define a reflective version of the λ-calculus, as a vehicle in terms of which to explain reflection to Jon Barwise.[37] I have therefore omitted the parenthetical from this version.

The motivation for the parenthetical remark is clear enough from the surrounding text. At the time I viewed the λ-calculus as fundamentally a declarative language for denoting functions, not as a procedural calculus. But on reflection I am not sure that is entirely correct. α and β-reduction are deeply enmeshed in the definition of the λ-calculus, and although there is no requirement that terms be reduced, and the Church-Rosser theorem allows one to side-step issues of reduction order, and so forth, I believe that our terminology is corrent: we are right to call the λ-calculus a *calculus*, not simply a *language*. That is: the λ-calculus is more implicitly procedural than is suggested by its inclusion in a list of "exemplars of the declarative tradition."

A83 ·78/0/2 «The following overlaps with annotation A13 in the POPL paper. Combine into one (the POPL version is better, except that I should include the "not pass functions upwards" comment from this one), and then simply have one annotation refer to the other.»

In a colloquium in the Artificial Intelligence Laboratory at SRI

philosophical insight that can be wrested from intensive engagement in the

International, in the spring of 1982, I gave one of the very first talks on 3Lisp. As it happened, John McCarthy (inventor of Lisp, and designer of Lisp 1.5) attended. Though as a young student I was almost paralytically anxious about making this claim in front of the great master, I nevertheless—not really having any other options—proceeded with what I had planned to say, and claimed that, according to my analysis, traditional Lisp's dynamic scoping protocols were a "mistake," to which quotation and other metastructural manoeuvrings were a partial and rather awkward work-around. In particular, they supplied "half" of the benefits of a true higher-order language, by providing a way of handing closures *downwards*, though there was no way to pass them *upwards* ("upwards" and "downwards" in terms of the usual notion of a control stack; this has nothing to do with levels in the reflective hierarchy).

I was surprised—and enormously relieved—that McCarthy very graciously, if laconically, agreed.

A84 ·83/0/1:2 More details are given in (the present volume's) ch. 4.

A85 ·84/-4/-3 Though, as stated here, I originally defined closures to be pairs, by the time the popl paper was written ("Reflection and Semantics in Lisp," included here as ch. 4) I had given them their own distinct structural category. There is merit in both approaches.[38] (For consistency, I have adopted the popl approach, in which they are their own distinct category, in figure 17.)

A86 ·86/-2/-5:-4 Re MacLISP see annotation A104, below.

A87 ·89/1 While it is correct that LAMBDA has first and foremost to do with naming, I do not believe that this paragraph is strictly correct. For a better analysis see dissertation section 4·c·i, included here as ch. 3.c.

A88 ·90/b2/-3:-1 «Ref Quine: from "On What There Is," *Review of Metaphysics… …*;included in *From a Logical Point of View* (Harper & Row, New York: 1953)»

A89 ·91/1/-2:-1 Cf. A79 (re ·99//1), above.

A90 ·91/-1/-7:-5 This claim is extremely important: that upon reflecting:

1. What is *used* prior to reflection is *mentioned*; and
2. What is *tacit* prior to reflection becomes *used* (not just *reified*).

details of a computational system

37. Cf. annotation A41 in ch. 4, particularly note 10.

38. On the one hand, closures, like all other structural categories except atoms (variables) and pairs (redexes), are normal form. On the other hand, like atoms and pairs, and unlike the other normal form categories, closures are

Not only did these points influence the approach to real-world ontology sketched in 03; they also infected the ideas I was mulling on, at the time, about fusing higher-order and intensional "objectification" levels in Mantiq (cf. A1).[38.5]

I still believe that a substantial issue remains lurking here, with which a proper theory of cognition should come to grips: relations between and among processes of

1. *Reification*—leading us to find the world intelligible in terms of objects;
2. *Semantic ascent*—generating quotation, meta-level concepts and expressions, and other forms of symbolic or cognitive "mention");
3. The use of *higher-order* structures (such as higher-order functions); and
4. *Reflection*—what we might call "*procedural ascent and descent*," involving all the issues adumbrated here, about stepping back, detachment, vantage point, etc.

In our formal efforts to be rigorously clear about the *differences* among these notions, we sometimes fail to recognize their *similarity*—and more seriously, what may be their common genealogy.

Note that Friedman and Wand's "Reification: Reflection Without Metaphysics," a paper I cite as indicative of the general reaction to 3Lisp, which set its semantical approach aside, can be understood in this light to be an attempt to wrestle with the first issue on this list without addressing the other three.

A91·92/0/-3:-1 It remains true, more than three decades later, that we do not yet have a semantical account of procedural reflection—and thus, by my lights, an adequate computational account of it either. This fact undoubtedly played some part in the decision to annotate and publish these papers now.

A92·92/1/-4:-3 The ability to supply the continuation to user programs is perhaps the one characteristic of 3Lisp that was recognized and provided in subsequently proposed reflective languages, such as Friedman's Brown.

A93 ·93/0 Cf. the discussion of the relation between the tower and levevl-shifting views in annotation A80, above.

A94·93/0/-3:-1 Cf. (this volume's) ch. 5.

A95·94/n16/4 The project to develop Common Lisp arose at an ARPA "Lisp Community Meeting" in April 1981. The language has had a long and rich history since then. See Steele & Gabriel (1993).

A96 ·95/4 This list should have a fifth entry, on studies of *self-knowledge* (cf. §... of ch. 2).

A97·97/-1/-9:-6 That *declarative semantics does not cross implementation boundaries* is an extraordinarily serious issue, which has yet to be theorized.

Suppose that architecture or virtual machine Y is implemented on top of language or system X. The question has to do with which of various properties P_i exemplified by X (the underlying system) are "inherited" by—i.e., true of—system Y, in virtue of the implementation relation holding between them. The answers are both complex and illuminating. There is no way that Y can be a "real-time" system, for example (in the sense of providing metric guarantees about certain kinds of behavior, such as providing support for a routine to run exactly once per second), unless X is also real-time. So, to adopt a convenient way of speaking, I would say that "being real-time" *crosses implementation boundaries downwards*. That is: from a system's being real-time—i.e, providing behaviour subject to metrical, not merely topological, time constraints—one can conclude that the system on or in which it is implemented is also real-time, and hence that the same is true of all such systems below it, down to and including the hardware. Conversely, "being a finite state machine" is a property that crosses implementation boundaries *upwards*, since there is no way to implement a machine with an indefinitely unbounded store on top of one that has no such store. Needless to say, being a finite state machine does not cross implementation boundaries *downwards*; you can perfectly well implement a finite state machine in Lisp, which is not one.

The present point is that declarative semantical properties in general—and thus reflection in particular, since it is defined in terms

not unique designators.

38.5. Cf. ch. 2, §....

38.7. A standard excercise that I assign, in introductory courses on the philosophy of computing, is to ask students to explain, for each item in a list of properties, whether or not they "cross implementation boundaries" upwards and/or downwards—not just *being real-time* and *being a finite state machine*, discussed in the text, but also: *being digital*, *supporting recursion*,

of declarative semantics—*do not cross implementation boundaries in either direction*. From neither X nor Y's being reflective, in the above example, can one deduce anything about whether the other is reflective.

For further discussion see comments in § «?» of the Introduction, and AOS.[38.7]

A98 ·99/-3 As discussed in ch. 2, I believe it is fair to say that the hopes expressed in this paragraph were entirely in vain. Dan Friedman, of Indiana University, was one of the most enthusiastic proponents of reflection in the programming language community; I owe him a tremendous debt of gratitude for the enthusiasm and support he offered subsequent to the publication of the POPL paper introducing 3Lisp ("Reflection and Semantics in Lisp," included here as ch. 4). However as perhaps best illustrated in his own paper with Mitchell Wand,[39] the first thing that most people did, in bringing reflection into their own work, was to dismiss every one of these six claims.

For some of the reasons for this dismissal see the discussions both in ch. 2 and in the Introduction. Fundamentally, I believe that it stems from the confluence of two issues: (i) the lack of appropriately strong theoretical engagement discussed in §1 of the Introduction between and among philosophical enterprises (philosophical logic, philosophy of language, philosophy of mind), the formal representational tradition (mathematical logic, computer science data bases, the AI and knowledge representation communities, etc.), and the programming language community; and (ii) the inadequacy of extant theoretical frameworks in all of these fields—inadequacies that I expect will not be overcome until the boundaries between and among them grow more permeable.

A99 ·100/1/-7 See also ch. 6.

A100 ·101/1/3 This statement remains true: I *still* intend to develop a reflective descriptive system. Cf. the discussion of the fan calculus in §6 of the Introduction, and of both the fan calculus and of Mantiq in ch. 2.

A101 ·103/1 This paragraph contains glimmers of the intuitions that will form the basis of the proposed fan-calculus, discussed in §6 of the Introduction, and in ch. 2.

It is no accident that this issue, and the importance of relating

being continuous (analoge) *vs. being discrete* (analog), *being correct*, *being physical*, etc.—including, not least, *being computational*.

39. "Reification: Reflection Without Metaphysics" (Friedman & Wand, 1984).

to the world as a whole, constitute the final substantive sentences in the chapter.

A102·103/1/7:13 These two sentences prefigure the issue that in the Introduction I label *blanket mechanism*:[39.5] an approach that what I take to be the most important set of considerations affecting computational architectures—declarative semantics and normative deference—in favour of a purely internal and purely causal account of how things work.

A103 ·104/0 It was not until 1987 that Rodney Brooks made his famous statement that the "representation" should be discarded in Artificial Intelligence systems—in favour of a view that, in his words, treated "the world as its own best model";[40] see also his "Intelligence Without Reason" and "Intelligence Without Representation."[41] What I take to be significant about the widely-heralded "sea-change" ushered in by the work of Brooks and others[42] is the fact that it betrays what I am here attributing to Weyhrauch: a somehow tacit but deep assumption that "representation" meant constructing within the machine *a replica of the world as a whole* that could be used in its place—as opposed to what cognitive scientists and philosophers of mind take a representational theory of mind to involve, which is that a person "represents" the world only in the sense of employing some interpreted symbols or structures with semantic content involving facts, entities, and states of affairs in the world. Even an internal structure with content along the lines of "Make sure you look out constantly and check the intersection to make sure that it is empty" would count as a representation on the latter view, but apparently not the former.

It is hardly surprising that the "full simulation" view of representation needed to be eschewed—though to take that as a rejection of representation altogether is both a rather extreme (and even binaristic) reaction. Brooks later softened his view, saying that systems should use representation "only when necessary"—which opens the door to what representation had originally meant.

For more on Brooks and on the circumstances in which, in my view, representation is required, etc., see "Rehabilitating Representation," ch. 7 of Volume II.

A104·105/-1/-5:-1 This dissertation was written in 1981 in Bravo, the first

39.5. See especially §....

40. Brooks 1987.

"WYSIWYG" ("what you see is what you get") document preparation system, implemented on the Xerox "Alto" minicomputer—arguably the first personal computer ever built, developed in the early 1970s at the Xerox Palo Alto Research Center (PARC). The first 3Lisp implementation was developed in MacLISP, a dialect of Lisp implemented under "ITS" ("Incompatible Time-Sharing System") at the Artificial Intelligence Laboratory at MIT, running on Digital Equipment Corporation PDP-6 and PDP-10.

A105.106/0/-1 Sure enough, the implementation listed in the appendix to the dissertation did contain a bug—and a serious one, at that. Though it handled reflective procedures correctly, and in general constructed, passed around, and called continuations appropriately, it failed to deal correctly with the rare case of *reflective continuations called in the course of normally processed code* (which, from a level-shifting point of view, require an instantaneous double level shift). That bug was corrected in the implementation presented in "Implementation of Procedurally Reflective Languages," included here as ch. 5.