# DCal — Register — 001

Author: Brian Cantwell Smith
URL: http://…

Posted: October 26, 2008
Last edited: October 26, 2008

## I • Prefatory

A. I have started working on the calculus to underlie Clapboard. This document is intended to be something like a blog or wiki entry. Until it is clear what term to use, I will call it the "DCal register." It is intended to be an occasional, running set of comments on issues, design decisions, opinion pieces, etc. I might try to put them on the server with blog software (MovableType, WordPress—thoughts?). But I don't want to wait on getting content out until that is decided.

B. For the moment, therefore, I'll use my standard notation/skeleton styles in Word, and post these as PDFs or something on the AOS/Clapboard wiki.

## II • DCal

A. I think we need (somewhat formally) to split the overall project into two parts:
   1. **Clapboard:** the coordinating infrastructure & system for discursive materials; and
   2. **Calculus:** the calculus on which I would like Clapboard—and a myriad other systems— to be built.

B. It seems to me that both projects should be able to end up being internationally networked, open-source projects, with contributions made by people and groups from far-flung places.
   1. The calculus will have to be a tighter, more focused effort; because it is not exactly (at least I don't currently imagine it as being) a system that supports plug-in extensions, etc.
   2. Clapboard, on the other hand, will very much be a system to which people contribute modules.

C. Calculus—name
   1. The problem with 'fan calculus' as a name is that it isn't easily pronounceable. FCAL etc., aren't very pleasing. If it is to fly, I think we have to have a *shorter* (max 2 syllable), *pronounceable* moniker,
   2. We've called the modules we've imagined for Clapboard "Dancing with … ". E.g.:
      a. Dancing with Documents (which I am working on)
      b. Dancing with Bibliographies (Jun & Alex Lamey)
      c. Dancing with Email (who is doing this?)
      d. … etc.
   3. That suggests we might use "dancer", or something, so that the "dancing with" modules would follow naturally. But some reasons against this:
      a. The "Dancing with … " modules are really Clapboard modules, not modules of the calculus per se;
      b. According to Jutta, 'Dancer' as the name of a programming language or system is taken (by people with McLuhan program ancestry, oddly enough).
   4. Instead, therefore, I have provisionally been using **DCal**, for "descriptive calculus". One could write this "decal", which is how it would be pronounced, though decals ("designs prepared on special paper for transfer to another material, such as glass, porcelain, or metal") isn't exactly a term of grace or gravity.

    5. So for now I will spell it "DCal—until or unless someone suggests something better.

D. Principles

    1. Below I will set out an initial (& provisional) list of 14 principles that DCal's design will be based on.
        a. They are deadeningly abstract. I'm not sure what that means.
        b. They are also unordered. They should be grouped in some way; but before trying to do that I want to let the sit a bit.
        c. There are also probably a bunch more than should be articulated. But they will be a start.
    2. Fans
        a. What's missing in the principles is any description at all of the **fan stuff (**fan-ins/fan-outs, abstraction, identity, etc.)—which was supposed to be the foundation of all of this.
        b. I'm not currently sure what to think
            i. Whether the fans generate these principles, if I could state them correctly;
            ii. Whether these principles generate the fans;
            iii. Or whether they are independent (which seems unlikely)
            iv. I guess what I think is that the fans are a possible way of *meeting* principle #1, of perspectival identity.
        c. There is also the question of whether—to caricature the disagreement between Steve and me—the issue of going "up & down" or "across" ties into this (i.e., should be a principle, or identified as a principled option, or something like that), or is a style of using DCal, or what.
        d. *Tai!*[1]

E. Perhaps my main conclusion, from starting to work on DCal, and spending some time articulating these principles, is how daunting this project is going to be. Fair enough; I shouldn't be surprised. But I've been sobered but how I don't have consistent vocabulary established (re content, meaning, use, interpretation,, etc.—though cf. my reactions to Israel & Perry's excessive neologisms for having certain parameters tied down). I don't yet know how the abstraction stuff is going to go. And so on. This is going to take work.

F. Still, I feel as if I can imagine DCal.

    1. When all is said and done, and if we do it right, I feel as if I will be able to say: "Yeah, that is exactly what I had in mind."
    2. And I won't be wrong. It is just a question of rendering it external & explicit …

G. So to the principles, in a moment. But first some comments on calculi in general (why this project is important).

## III • Calculi

A. Intro

    1. Foundational calculi have played a crucial role in the development of science.
    2. The differential calculus was critical to the development of physics; logic to the study of reasoning & entailment; algebra to the arithmeticisation of geometry; etc. They are not "empty vehicles", in other words; they tailor (shape, encourage, restrict) imagination in tremendously consequential ways.
    3. Getting a calculus "right" is hugely beneficial in allowing us to register appropriately, and

[1] My ubiquitous marginalia & note inscription for "think about it!"

thereby come to understand, diverse subject matters.[2]

4. What it is to be "right" (better or worse) is epistemologically & ontologically subtle.[3]

B. Ontic commitments
   1. In terms of ontic/metaphysical commitments—in how they register or allow registration of the world—calculi effectively make or impose a 3-way distinction among:[4]
      a. **Kernel** Content: What is embodied or "built in" to the calculus itself—and can therefore be assumed to be true (or at least claimed) or all subject domains described in it;
      b. **Constructed** Content: What is then "said" in the calculus—in specific descriptions, theories, and claims; and
      c. **Prohibited** Content: What is effectively "disappeared" or removed from the discussion, in virtue of falling outside (a) & (b), because it "cannot be said" (violates kernel assumptions in some way).
   2. One way to group these is to think of it as a 2-way distinction, between
      a. What can be said—which in turn breaks down into
         i. What is said *implicitly* or *automatically*, as it were, in virtue of the calculus' inherent structure—i.e., category (a); and
         ii. What is said *explicitly*—category (b);
      b. What cannot be said—category (c)[5]
   3. Discussion
      a. What is "built in" will typically be embodied in (i) **syncategorematic** structures & operations, and/or (ii) various **primitive** symbols & operators. We call such characteristics **kernel** (rather than primitive or syntactic), and thus speak of the **kernel (ontological) commitments** embodied in its structure.
      b. Thus the differential calculus makes a kernel commitment to the fact that the regularities it can be used to express will be formulated as derivatives & integrals of—usually temporally—dependent measure variables. Even purely mathematical calculi without obvious concrete subject matters, such as algebra & set theory, still typically embody specific ad/or particular kernel commitments.
      c. What is "said" in the calculus—i.e., the "meaning" or "content" of its descriptions—we will call **constructed commitments**.
      d. The net registration of the world embodied in a set of DCal structures will consist of its kernel & constructed commitments.[6]

[2]Cf. Newton's early work on a calculus based on the *radius of curvature* of a function, at a given value, as a basis in terms of which to frame the laws of motion—a project that didn't work out very well. The shift to the less-geometrically evident notion sof *slope* was radically more congenial to the framing of the world's physical regularities.

[3]It of course depends on the purposes for which the calculus is needed or used.

[4]Calculi are thus a kind of language, though I make no claim here as to *what* kind. Among other things, calculi are clearly more "formal" than natural languages, implying that the divide between "the language itself" (its kernel commitments) and "what is said in the language" (its constructive commitments) is sharper than in the natural case. Note, however, that as with everything, what constitutes "DCal itself" versus an instantiation of DCal, extended with various constructed structures & commitments, is not an intrinsic matter; what is the case will depend on how the various systems are respectively registered (including the denotation of the name 'DCal', which again is not fixed by the system's design).

[5]Cf. the "impossible zone" in Haugeland's "Truth & Rule-Following."

[6]As a teenager, I was intrigued by the division of labour between the design of the differential calculus and the formulation of the laws of motion within it. As a freshman, I asked my physical professor what fraction of Newton's brilliance he would allocate to the two aspects of the development of classical mechanics & dynamics. He asked what I thought; and I said "90% for the calculus; 10% for the laws of motion." Not really a serious remark, of course; but the intrigue remains, and I might stand by the higher ranking of the formulation of the calculus.

C. Design
   1. Common use of a calculus helps one compare and contrast divergent claims or registrations expressed within it.
   2. Sciences (such as contemporary syntactical linguistics) in which there is not a common calculus, with each theory then being expressed in its own formalism, make such comparisons vexatiously difficult.
   3. I take judicious *allocation of ontic commitments across the kernel/constructed divide* to be one of the most important normative criteria on a calculus' worth. Excessively general calculi (with little kernel structure & commitment) provide the theorist with no help in registering the world. Conversely, calculi can constrain imagination to regularities expressible in their terms; theories or suggestions that violate their kernel commitments can be difficult to communicate or express, often leading to misunderstanding.[7]

D. Examples
   1. The following 5 calculi are (±) among the most important to have been developed to date:
      a. **Algebra**
      b. **Differential calculus** (built on top of algebra)
      c. **Set theory**
      d. **λ-calculus**
      e. **Formal logic** (propositional, predicate & quantificational)
   2. Some mathematical formalism & systems (such as dynamical systems theory [DST]) receive a lot of development for use as a framework in terms of which to register phenomena, but aren't themselves calculi (DST uses algebra & the differential calculus)
   3. Another *set* of calculi—or anyway they might be considered as such—are the raft of programming languages that have been defined over the last several decades. They are systemically different, however, in that what is (formally) defined is *what happens*, rather than *what they mean*.[8]

## IV • Mandate

A. Why do we need a new calculus?

B. Oddly enough, I think I am so close to the situation that I am not yet in the best position to formulate an answer. But my basic sense is that no calculus, to date, has dealt seriously with *objects* or with *description*.
   1. Algebra and the differential calculus deal exclusively with measure variables;
   2. Formal logic deals with propositions, and in predicate and quantificational form, with predicate holding of objects. But objects are dealt with as unproblematic, singular unities, preëmptively registered.
   3. The objects that computational systems have to deal with aren't idealized singular unities; they are complex abstractions with textured and interlocking identity conditions.

C. Note: When I abandoned the development of 4-Lisp, in the late 1980s, I did so because I did not know how to deal with real-world objects (the original topic of the Mantiq project; 3-Lisp was just intended to be a design study en route to Mantiq). One could argue (with only a modi-

[7] In this sense calculi (and perhaps all languages) establish a particularly simple typology of the "domain of comprehensibility" within which the possible & the actual can be distinguished from the impossible but conceivable, as opposed in term to the inconceivable.

[8] The term "meaning" has been appropriated for programming languages, and put into service as a way of indicating what happens. Programming language semantics, however, is not *semantics*, in my book. So I stand by my claim.

cum of retrospective guile) that the Objects book (O3) was the outcome of a long digression to figure out what objects were. So in a sense, this whole DCal project could be viewed as 4Lisp redux—though with description, rather than execution, at its heart.

D. Perhaps the best thing to say for now, however, is that, in my judgment, computational systems, or a computationally mediated life, requires a descriptive calculus that meets the 14 principles identified below.

## V • Principles

A. DCal is a *calculus of description*, designed to satisfy a dozen passel of fundamental principles:

| | **Property** | **Description** |
|---|---|---|
| **P1)** | *Perspectival identity* | Identity is not taken to be an intrinsic property of anything (including DCal structures themselves). Rather, descriptions that depend on issues of identity—of property & type as well as object or individual—must "apply" individuation criteria as part of their meaning or content. The issue of whether that which is registered "satisfies" the relevant identity criteria is part of what determines how & whether the description "fits" the world—meaning that DCal descriptions & terms, like sentences in traditional calculi, have "success conditions." |
| **P2)** | *Deferential semantics* | in a very broad sense, DCal structures are reminiscent of *representations,* in that containing or conveying information *about something else*, rather than (except in extremely rare cases) *themselves* being that of which they speak. We say that DCal descriptions *register* their subject matters.[9] Although registrations, including how they are used, shoulder responsibility for (i.e., are the locus of the determination of) how they register their subject matters, and although normative considerations that stem from this use, it is nevertheless presumed that *it is the world* (i.e., that which they register) *that is the truth maker*. In this sense of being normatively *deferential* to the world the semantics has a classical flavour. |
| **P3)** | *Contextual registration* | Descriptions are taken to be arbitrarily *contextual* (deictic/indexical, relative to conceptual scheme, instant of use, etc.,) at arbitrary scale—not just "within sentences" (or other complexes).[10] It would thus be natural for a DCal system to have structures analogous to such English phrases as *I*, *you*, *my*, *today*, *local*, *John*, *recently*, *this*, *that*, etc. |
| **P4)** | *Dynamic registration* | DCal descriptions can not only be used to register temporal phenomena (i.e., be dynam*ical*) but can *themselves* be temporal (i.e., dynam*ic*). Cf. not only clocks, meters, sundials, etc., but rhythmical patterns, oscillations, movements, etc. |
| **P5)** | *Non-conceptual content* | While some descriptions may register their subject matters in terms of "classical ontology" (objects exemplifying properties, standing in relations, grouped in sets, and arrayed in states of affairs) DCal is not itself committed to such registration, and supports others as well (such as Strawsonian "feature-placing," to say nothing of measure variables as in the differential calculus). |

[9]No DCal structure, therefore, will *be* the name of a book, or the length of a list, or the address of a cell (though there may be structures that register that name, length, & address in canonical (normal-form) ways.

[10]Cf. Frege

| | Property | Description |
|---|---|---|
| **P6)** | *Metaphysical Holism* · | Rather than assume that the world is assembled from atomic or elemental parts, the background metaphysical assumption underlying DCal semantics is that the world is aboriginally *whole*, and that descriptions register *parts* of it under normatively-governed purposes of abstraction.[11] |
| **P7)** | *Meaning as (Partially) Use* · | It is not a kernel assumption that descriptions register *independently* of how they are used, nor that their significance derives *wholly* from how they are used. Rather, use is (in general) viewed as a *partial determinate* of meaning. |
| **P8)** | *Registration* · | It is traditional to view reasoning as a challenge of selecting and carrying out an appropriate (perhaps complex) series of inferential steps based on a presumed, classical ontology (cf. P5)—i.e., as determining an arrangement, given a basic set of building blocks (or puzzles pieces). DCal is founded on a different view, which views the determination of an appropriate registration scheme as an equally (if not more important) step, with the reasoning "in that scheme" as simpler. |
| **P9)** | *Reflection* · | DCal is reflective as well as recursive, giving the user unprecedented control over the structure, operation and interpretation semantics of all described (constructed & kernel) structures. A kernel mechanism is provided with which to refer to or "mention" DCal structures, operations & interpretations—though what exactly is thereby mentioned (type, token, meaning, use, etc.) depends on how it is registered. With these reflective capacities, DCal structures, operations & interpretations can be overridden at will, providing that such overriding can itself (ultimately) be described in kernel terms. |
| **P10)** | *Fusion* · | The DCal structural field will appear (implicitly) to *fuse,* as much as possible, structures that "mean" the same thing with respect to the concepts & types in terms of which they register their subject matters. More precisely: it will be a normative criterion on the structural field to support registering structures with respect to identity criteria based on (various kinds of) meaning. ⟦Does this follow from P1?⟧ |
| **P11)** | *Formality* · | In spite of being a well-defined computational calculus, DCal is intended to be thoroughly "non-formal" under a variety of meanings of that term. Any attempt to develop a set theoretically based model theory for a system implemented in DCal (i.e., for a DCal system with substantial constructed content), or to prove its fundamental soundness &/or completeness, will be based on a misunderstanding. |
| **P12)** | *Interpretation* · | It is traditional to view formal calculi as "uninterpreted" systems of marks, with issues of semantic interpretation left outside the realm of the calculus per se, although in different calculi the kernel operations are typically defined with respect to (something like) a specific interpretation or interpretation schema (formal logic being the most extreme, in some peoples' minds challenging its claim even to be a calculus because of its semantic commitments). DCal, in contrast, includes (as a kernel constituent) a DCal account of its own interpretation, in terms of which kernel operations are defined and reflective facilities described. As much as (effably) possible, that is, DCal is intended to *embody* a particular ontological/metaphysical view. |

[11] Except that 'abstraction' is not used to mean what it classically means.

| | Property | Description |
|---|---|---|
| **P13)** | *Differentiation & Abstraction* · | DCal's approach to identity is based on a "fan-in/fan-out" conception of (something like[12]) *abstraction*, in which regions of the world are gathered together and taken as unities or singularities for some purposes, and divided into pluralities or registered in other ways for others. Notions of sets vs. their members, parts vs. their wholes, abstract entities vs. their concrete exemplars, types vs. their tokens or instances, etc., are all characterized as "differentiations" of the basic fan-out model. That is: distinctions between and among sets & members, parts & wholes, abstract & concrete, types & tokens & instances, etc., will all be matters of constructed, not kernel, content. |
| **P14)** | *Physicality* · | Notions of locality, accessibility, etc. in DCal (i.e., those relations that can lead to things happening in unit time) are based on concrete, physical connectivity & connection via effective properties. There is no notion of *syntax*, per se, but rather of (spatiotemporal) concrete immediacy. |

## VI • Postscript on Notation

A. A final note on the general issue of notation.

B. Lisp
   1. I was always a kind of fan of Lisp's stunningly simple lexical syntax (with caveats about the identity of lexical variables—more on that at another time). But lots of people hated it. It was commonly said that 'Lisp' stood for "lots of irritating parentheses".
   2. Some merits of Lisp syntax (imho):
      a. It is dead simple (you can teach it in its entirety in about 20 minutes)
      b. It is trivial for reflection, because there are no complex syntactic types
      c. It can be typed in ASCII, without requiting a GUI interface
   3. Some demerits:
      a. Scoping, though explicitly marked, isn't psychologically obvious; it can only be easily conveyed with proper "pretty-printing" (i.e., indenting)
      b. Even automatically balancing parentheses tricks in editors only partially mitigate the difficulty.

C. A possible way to improve the situation is the following:
   1. Although we all primarily use *character–based input* (keyboards, ability to incorporate in text files, email messages, etc.), we also all primarily (virtually universally) use *bit-mapped output*.
   2. If there could be a better way to print structures out, therefore, so that they could be easily and transparently read, which relied on bitmapped displays, in a way that could still be used by appropriate editors for real-time display and editing, but which didn't require non-ASCIi channels for input, that might ease the situation.

D. I designed a trial version of such a thing (for Lisp; not for DCal), demonstrated below in a side-by-side comparison of the 3Lisp reflective interpreter code in the old (lots of parentheses) and new (boxed) notation. I've started to articulate some rules for display that would "print out" the structure lexically indicated on the left into the graphical structure shown on the right—but they

[12] Only "something like" because it is classically assumed that "abstract" individuals are not concrete, whereas in DCal ontology/metaphysics, all individuals are based on an act of abstraction. Because what is registered is not the "abstraction," but that which is gathered together as a unity, there is no lack of concreteness in the "abstracted" individual.

are pretty obvious.[13]

E. Immediate reaction
   1. My sense is that, for the boxed notation to be usable, the grid lines—like grid & guide lines in Illustrator, InDesign, and similar graphics programs—need to be very faint. Otherwise they demand attention and chop the text up far too much.
   2. On the other hand, the notation does seem to me to accomplish one goal: immediate visual accessibility of scope, without a lot of fancy characters or character patterns (as in HTML).
   3. What do you think?

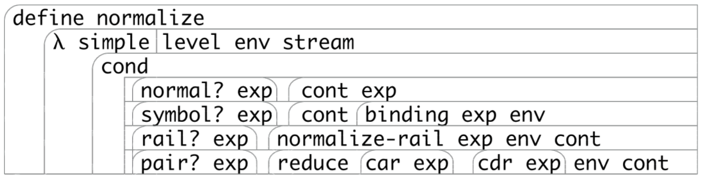
F. Comparison

## 3Lisp Reflective Processor

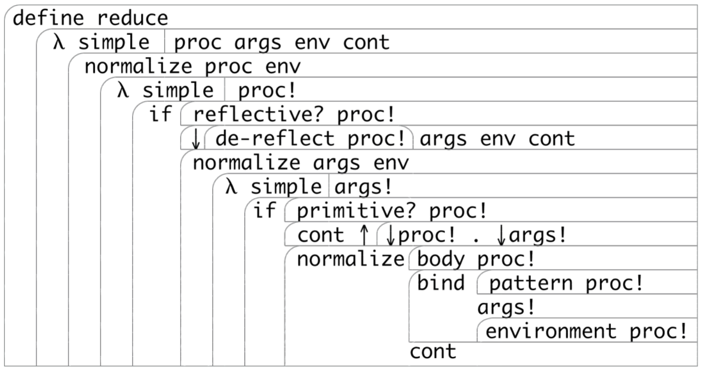
**Standard Notation**

```
(define normalize
   (λ simple [exp env cont]
      (cond [(normal? exp) (cont exp)]
            [(symbol? exp) (cont (binding exp env))]
            [(rail? exp) (normalize-rail exp env cont)]
            [(pair? exp) (reduce (car exp) (cdr exp) env cont)]])))

(define reduce
   (λ simple [proc args env cont]
      (normalize proc env
         (λ simple [proc!]
            (if (reflective? proc!)
                (↓(de-reflect proc!) args env cont)
                (normalize args env
                   (λ simple [args!]
                      (if (primitive? proc!)
                          (cont ↑(↓proc!.↓args!))
                          (normalize (body proc!)
                                     (bind (pattern proc!)
                                           args!
                                           (environment proc!))
                                     cont)))))))))

(define normalize-rail
   (λ simple [rail env cont]
      (if (empty? rail)
          (cont (rcons))
          (normalize (1st rail) env
             (λ simple [first!]
                (normalize-rail (rest rail) env
                   (λ simple [rest!]
                      (cont (prep first! rest!))))))))))

(define read-normalize-print
   (l simple [level env stream]
      (normalize (prompt&read level stream) env
         (l simple [result]
            (begin (prompt&reply result level stream)
                   (read-normalize-print level env stream))))))
```

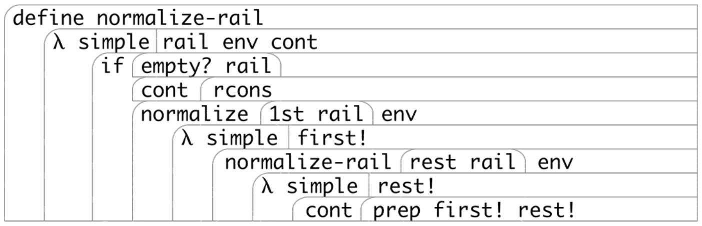
**Box Notation**



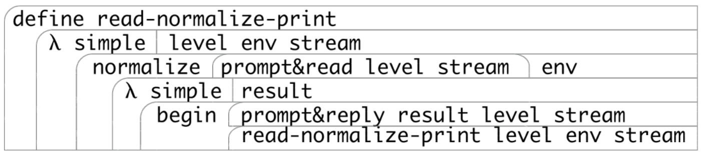
```
define normalize
   λ simple level env stream
      cond
         normal? exp   cont exp
         symbol? exp   cont binding exp env
         rail? exp   normalize-rail exp env cont
         pair? exp   reduce car exp   cdr exp env cont
```



```
define reduce
   λ simple  proc args env cont
      normalize proc env
         λ simple  proc!
            if reflective? proc!
               ↓ de-reflect proc! args env cont
               normalize args env
                  λ simple args!
                     if primitive? proc!
                        cont ↑ ↓proc! . ↓args!
                        normalize body proc!
                                  bind  pattern proc!
                                        args!
                                        environment proc!
                                  cont
```



```
define normalize-rail
   λ simple rail env cont
      if empty? rail
         cont rcons
         normalize 1st rail env
            λ simple first!
               normalize-rail rest rail env
                  λ simple rest!
                     cont prep first! rest!
```



```
define read-normalize-print
   λ simple  level env stream
      normalize prompt&read level stream  env
         λ simple result
            begin prompt&reply result level stream
                  read-normalize-print level env stream
```

[13] For old-time Interlisp aficionados, this can be viewed as a 2-dimensional strong extension of Interlisp's use of ']' to end a lot of parenthesized structures.