**Logical Implication in predicate level LC Type Logic**

Despite the myth, the difference between Proplog and Predlog is not one of generality. Wherever there are variables there is generality, since a variable symbolizes a type of prototype, or a combination thereof, whose values are its instances. Indeed, without explicitly mentioning the quantifiers, a tautology shows that it holds true for *all* values of its variables, a contradiction for *none*, while all other well-formed schemata will be *not* true for *some* values.) What distinguishes Predlog, and justifies its claim to be an extension of Proplog, is its ability to draw inferences from assertions based on dependencies between *parts* of those propositions.

The specific "parts" Predlog is concerned with are the things we are talking about and the things we are saying about them, a general – if vague – distinction that goes back at least to Aristotle. While traditionally the distinction was drawn in terms of grammar, the formalizing of Predlog (following Frege's symbolic notation for these parts) convinced some logicians that grammar was by itself, an inadequate criterion. Already in Russell, a distinction was drawn between the "grammatical" and the "logical" division of subjects and predicates. But Russell's formalism still forced him to consider only signs by themselves (that is by redistributing their grammatical roles in a manner abstracted from semantics or usage). The LC insight is ( to paraphrase Wittgenstein) that meaning is determined by use – only in some usage/context can we know how to assign logical subject (i.e., "Locator") and logical predicate (i.e., "content"). (only in context can we determine if "wise" is predicated of "Socrates" or vice versa.) And the least ambiguous usage is that in which a proposition figures as the answer to a query. (this too traces at

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

least to Aristotle, for whom the "dialectic" was a formal question-and-answer exercise for correctly parsing propostions.)

*Formation Rules*

I. If $C$ is the free variable of $n$ bound variables of a well-formed query, and if $\ell_1, \ell_2, \ldots, \ell_n$ are the bound variables, then $C(\ell_1, \ell_2, \ldots \ell_n)$ is well-formed.

II. If $C(\ell)$ is well-formed, then N $(C(\ell))$ is well-formed. (And NN$(C(\ell))$, etc.)

Cor. 1. If $C(\ell)$ is well-formed $\sim C(\ell)$ is well-formed.

Def. 1.) If $C(\ell) = fx$, N$(C(\ell)) = (x) \sim fx \equiv^{n} \sim p\&\sim q\&\sim r \ldots$ for all values of $fx$

Def. 2.) If $C(\ell) = fx$, NN$(C(\ell)) = (*x)fx \equiv pVqVr \ldots$ for all values of $fx$

Def. 3.) If $\sim C(\ell) = \sim fx$, N$(\sim C(\ell)) = (x)\, fx \equiv p\&q\&r \ldots$ for all values of $fx$

Def. 4.) If $\sim C(\ell) = \sim fx$, NN$(\sim C(\ell)) = (x)\, fx \equiv \sim p\mathrm{V}\sim q\mathrm{V}\sim r \ldots$ for all values of $fx$

The quantifiers are the additional "logical constants" of the Predlog calculus. They signify, like the connectives of Proplog, ways of combining propositions into truth-functions. And like the connectives of Proplog, the significance of deriving them from the set-negation operator is 1.) to show their interdependent relations to the proposition/propositional functions from which they are produced. (As in the interdependency between number-systems and their operators: the calculus in which they are combined is what counts; or to paraphrase

[n] '≡' again implies a t-functional equivalence.

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

*TLP* (5.44, 5.5151), the possibility of one is "written into" (i.e. presupposed) the other.. The operation that connects the propositional function and the quantified proposition as base and result shows that the relation between them is internal.

But likewise, 2.) the same interdefinable relations held between the connectives of Proplog (and highlighted by their t-functional derivation from N(*)) also apply to the quantifiers: they can be defined in terms of each other; which of course implies that the Predlog notation doesn't requires both the existential and universal quantifiers – one can be reduced to the other. (For the purposes of this paper I'll give preference to the universal form, only giving the existential when its semantics lends itself to an easier understanding). Here are the t-functional equivalencies:

$(x)fx \equiv \sim (*x)\sim fx$

$\sim(x)fx \equiv (*x)\sim fx$

$(x)\sim fx \equiv \sim (*x)fx$

$\sim(x)\sim fx \equiv (*x)fx$ [read, eg., "not everything is not red" for "something is red."]

Now we can support some simple LC dialectics:

I. The query Stores. *~Sales corresponds to the Predlog assertion $(x)f$x for the two domains 'Store' and 'Sales,' or in English, 'For all values ("instances") of *x* in the domain (= "of Type") 'Store,' the value of 'Sales' is *f*. Units can be defined as a constraint on the domain ("sales *in dollars* is *f*"). Again, *variable* in Predlog indicates *Type*, *value* indicates instance. And again, the assignment of which Type to which variable (*f*, for Content; *x* for Locator) is determined by the form of the query. (Had 'Stores' been the free variable, and 'Sales' the bound, the substitutes would have been switched). Notice the content variable *f* is kept open in the assertion: as I mentioned

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

earlier, it is only the *heading* of a relation that corresponds to a Predlog assertion; the *body*, the tuple rows asserting the instance-pairs of Stores to Sales, corresponds to their Proplog (moleculas) conjunction (in this case, on (utomic) proposition per tuple). Traditionally in logic the *f* is construed as a simple predicate common to a number of subjects: '*x* is red.' But in a database relation each content-instance logical (predicate) may be different, and only the content-Type is shared. This not necessarily a drawback, as some logicians have always treated the *f* as an open (free) variable: since in the Predlog notation the only alternative is the to treat it as "closed," i.e., bound by a quantifier; and since bound predicate/function variables implicate second order Predlog, which was, among other concerns, associated with the logical paradoxes. However, many are troubled by the free form. We have already seen how Quine treated the *f* as a "schematic letter" standing for a predicate or property; but how this "standing for" differs from a variable's is less than clear. It is also likely to have been considerations along these lines that led to Codd's removing predicate variables from the column-headers (that is, in addition to the second order concerns mentioned earlier).

In order to take a stance consistent with the variant usages, then, I would suggest the following: the *f* or content variable in a first order expression is a free variable open to a range of interpretations/scopings: from a single content value absent any reference to type[n], to a variable ranging over a type as domain (as in the example above) to one indicating no more no less than the formal concept "Content," i.e., the prototype itself, as in the uninterpreted schemata of Predlog. Moreover, for logicians *uncomfortable with the*

[n] That is, in cases where the content value is globally unique, or asserted without reference to type. In ordinary discourse, the statement 'Socrates is wise' (assuming context clarifies that 'wise' is the content) may be used meaningfully without explicit type reference (say, "degree of intelligence/knowledge" or some such. (If asked, how do you know he is wise, the Athenian might answer, "because the Delphic Oracle said so," or, "because he says he isn't," without having to specify type.

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

*presence of a free* variable in a proposition, we could interpret the above as stating "for all instances *x* of Store, there is some Sales value *f*." Of course, if pressed I'd have to confess that this interpretation sounds a lot like a second order assertion, namely '(*x*)(**f*)*fx*'; but then, LC-log does not incur the paradoxical consequences associated with second order Predlog[m]. And of course this Predlog assertion (and its corresponding relation) would be made *false* for a missing store; but again LC-Log can handle this.[n]

II. So the query (Store.* X Time.*)~Sales corresponds to the Predlog assertion '(*x*)(*y*)*f*(*x*,*y*).' The first thing to note is that the Predlog use of the universal quantifiers does not require the 'X'; the Cartesian product is already implied by the form. (To see this, we can first substitute for the values of *x*, which yields the expansion (*y*)*f* (a, *y*)&(*y*)*f* (*b*, *y*)&(*y*)*f* (c, *y*) …, which substituting for *y* yields *f*(a,a) & *f*(a,b)& *f*(a,c) …& *f*(b,a)& *f*(b,b) & *f*(b,c) … & *f* (c,a) & *f*(c,b) & *f*(c,c) … etc. And the second thing to note is that this form of expression is made false for a false or missing content-value, but meaningless for a false of missing location-value.

III. The query '(Store.* X Time.* X Product.*)~Sales, Cost, Profit' corresponds to the Predlog (x)(y)(z) *f* (*x*,*y*,*z*)& *g*(*x*,*y*,*z*)&*h*(*x*,*y*,*z*)'; which yields a relation (/cube) that looks identical to the RM relation we saw earlier: except that *different* inferences can be drawn from this Predlog form than the RM Predlog expression we saw, i.e., '(*x*)(*y*)(*z*)(*u*)(*v*)(*w*) f (*x*,*y*,*z*,*u*,*v*,*w*)'. Consider the following Predlog inference schemata [simplified to two locators (say, Stores X Time) and two contents (say, Sales, Cost).]

(a) (*x*)(*y*) [*f* (*x*, y) & *g*(*x*, *y*)]

(*x*)(*y*) *fx*

[m] To show why not will have to wait until a further paper/chapter. If not, the answer can be found in "Appendix II" (1998).

[n] To show how will also have to wait. If not, the answer can be found in the 3-logic paper (1993/4).

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

$(x)(y)\ gx$

(b) $(x)(y)\ [f(x,y)\ \&\ g(x,y)$

$\underline{(*x)(*y)\ f(x,y)}$

$(*x)(*y)\ g\ (x,y)$

(c) $(x)(y)\ [f(x,y)\ *\ g(x,y)]$

$\underline{(*x)(*y)\sim g(x,y)}$

$\sim(x)\sim(y)\ f\ (x,y),\ (*x)(*y)\sim f(x,y)$

None of these forms could have been inferred from the RM Predlog form of expression because in that form $[(x)(y)(z)(w)\ f(x,y,z,w)]$ there are not *two* contents, but one (i.e., *R*, for the entire relation). This can be more easily seen if we compare the (Proplog) bodies of the relations. In the RM form each tuple corresponds to a unique (atomic) proposition. In LC, each tuple corresponds to a (moleculas) conjunction of two (atomic) props ($p\&q$), asserting roughly Sales (Store, Time) & Cost (Store, Time). (And so in the simplified form there are twice as many propositions in LC, and in the fuller form (III.) three times as many props in LC as in the RM form.)

LC allows us to model dependencies *between* values of a tuple. We can think of the locator signs as the independent variables and the content signs as the dependent variables: as L varies, so varies C. To illustrate the different sores of inference possible, when we make the LC distinction, consider:

1.) $\underline{(*y)(x)\ f(x,y)}$ , and 2.) $(x)\ (fx\ *\ gx)$

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

$(x)(*y)\,f\,(x,y)$ $\quad\quad$ $\underline{\sim(*x)\,gx\_\_}$

$\sim(*x)\,fx$

Both are inference schemata of three logical variables, the first dyadic function, or one C of two L's; the second, two monadic functions, or two C's of one L. Let us say for 1.) x = Type (or domain) 'Customer,' *y* = Type 'Employee' and *f* = 'is client of' [We can easily imagine this as referencing a RM relation of two columns (Employee, Customer) and 'client of' – the external (absent) predicate – as the name of the entire relation.] What the premise says is 'Some employee has every customer as client,' and the conclusion (uninformative as it is) says 'Every customer is the client of some employee.' The difference between premise and conclusion point the different scopings of the quantifiers. In Predlog the outermost quantifier has the widest scope (and each successive quantifier a wider range than the succeeding). When the universal quantifier has an existential within its scope, as in the conclusion, it implies "some one or other, not necessarily the same one" ; and when the existential has a universal quantifier within its scope, as in the premise, it implies "some one and the same." (See Ambrose/Lazerowitz, p. 54.)

Now if the Type structure gives three instances of Customer, a, b, and c, and two instances of Employee, d and e, substituting for the premise first gives us $(x)\,f(x,\text{d})$ V $(x)\,f(x,\text{e})$, and then expands to $[f(\text{a,d}) \,\&\, f(\text{b,d}) \,\&\, f\,(\text{c,d})]$ V $[f\,(\text{a,e}) \,\&\, f\,(\text{b,e}) \,\&\, f\,(\text{c,e})]$. The conclusion becomes, first, $(*y)\,f(\text{a},y) \,\&\, (*y)\,f(\text{b},y) \,\&\, (*y)\,f\,(\text{c,y})$, and finally $f\,(\text{a,d})$ v f (a,e) & $f\,(\text{b,d})$ v f (b,e) & f (c,d) v f (c,e). We can then see, truth-functionally, how every condition that makes the disjunction of the premise true, makes the conclusion true: *and*, likewise, how the inference does not go the other way. (The example uses only the Types

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

and Instances for Customer and Employee: to find out which one, d or e, is the stellar employee of the premise we would have to consult the data.) Although this is a simple, and relatively useless inference, Predlog can be used to model arbitrarily complex relations [eg., adding to the above: Locator *z* in the domain 'Product,' and Content-functions *s* in the domain 'Store,' *c* in the domain 'Customer Category' (say, 'preferred' 'loyal,' etc.), and *p* in the domain 'purchase' (in, say, dollar units), we can write $(*y)(x)(*z)\ \{sy\ \&\ [(cx\ \&\ p(x,z))\ *\ f(x,y)]\}$. (You should have all the tools necessary to give the English equivalent (and hopefully the LC DBL Background schemas) of this … Hint: when the universal falls within the scope of an existential quantifier, it implies (though not necessarily) a l-N relation of the former to the latter.)

Example 2.) above gives a simple inference of one locator, two contents, say "For all mail-order customers *x*, if *x* resides in city *f*, *x*'s state is *g*; no customer mailings go to state g; therefore no *x* resides in *f*." [n] The first premise of 2.) is the first of the four Categorical Proposition-forms (traditionally used to solve syllogisms). Here they are in non-traditional form, that is by quantifiers-columns and equivalency-rows [the underlined expressions indicate the original A, E, I, O forms]:

A, universal affirmative $\underline{(x)\ (fx\ *\ gx)}\ \equiv\ \sim(*x)\ (fx\ \&\ \sim gx)\ \equiv\ \sim$ O

E, universal negative $(x)\ (fx\ *\ \sim gx)\ \equiv\ \underline{\sim(*x)(fx\ \&\ gx)}\ \equiv\ \sim$ I

I, particular affirmative $\sim\ (x)\ (fx\ *\ \sim gx)\ \equiv\ \underline{(*x)\ (fx\ \&\ gx)}\ \equiv\ \sim$ E

O, particular negative $\underline{\sim\ (x)\ (fx\ *\ gx)}\ \equiv\ (*x)\ (fx\ \&\ \sim gx)\ \equiv\ \sim$ A

[n] Understand, I am *not* implying (from the foregoing) that RM does not or cannot make these types of inference: it can easily do so *between* relations: that *is* what the algebra and calculus do, and do well. (Although, I must admit that this particular eg. $((x)\ fx\ *\ gx)$ involves the notions of "class inclusion," c, and type inheritance, relations that RM has more difficulty with than "union," u, or "intersection," n.) I *am* saying, however, that RM doesn't do this *within* a relation, which, by definition, contains only1 predicate (R itself).

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

The letters A and I are from the first two vowels of *Affirmato*, E and O from the two vowels of *Nego*, because traditionally the first and third were given by their affirmative and the second and fourth by their negative forms. The formalizing of Predlog *however gave* rise to complexities of inference that far out paced traditional syllogistic reasoning. (Ambrose/Lazerowitz give the following example (p. 46)):

All wild azaleas are easy to grow but not easy to transplant

Some things are easily transplanted

There are things which area either not wild or not azaleas

Whose form is given as:

$\{ (x) [fx \ \& \ gx * hx \ \& \sim jx] \ \& \ (*x) \ jx\} * (*x) (\sim fx \text{ v } \sim gx)$

Or consider the following inference schema:

$(x) \ fx * (\sim gx * hx)$

$(x) \ gx * (kx \ \& \ lx)$

$(x) \ (fx \ \& \sim hx) * lx$

The first premise suggests a relation of set ($fx$) to two mutually exclusive subsets ($gx$, $hx$), and the second premise can be thought of as a relation between a subset and its particular properties or constraints. In general, the move from whole to part involves a subset of instances, but a superset of properties or constraints: to borrow an example from Pate/Darwin, more constraints apply to squares than rectangles as a whole. Likewise the move from part to whole implies a superset of instances, but a subset of constraints.

Now let's consider multiple inheritance of properties; for instance, 'rectangle' and 'rhombus' are not mutually exclusive subsets of the set 'four-sided polygon' : (x) ($gx$ v $hx$) * $fx$. The vel (v) of logic allows for the truth-functional cases where $x$ (say, in the

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

domain ‘polygon’) is a ‘rectangle’ (*gx*), a ‘rhombus’ (*hx*), or both (from any of which we can infer it is ‘four-sided’ (*fx*)). And further, (*x*) (*gx* & *hx*) * *kx*; that is, when it is *both* we can conclude that *x* is a subset of *gx* and *hx*, and a sub-subset of *fx* (sub-sub-subset of *x*): it is a ‘square’ (*kx*)[n]. Squares have “multiple inheritance” of properties from rectangles (“right-angled”) and rhombi (“equilateral”) – they are a subset of each, but inherit all the defining properties/constraints from both.

It is here, however, that LC must begin to leave Predlog behind. The direct definition and structuring of hierarchies through operations, allows for direct inferencing in LC, along axes of resolution that may be many iterations deep; Whereas similar inferences in Predlog quickly become cumbersome. Consider a simple syllogism: ‘If all whales are mammals and all mammals vertebrates, then all whales are vertebrates.’ Although there is more than one way to express this in Predlog, I prefer the full-blown second order approach. (If second order offends you, ignore the opening quantifiers.):

(*f*)(*g*)(*h*){(*x*)(*fx* * *gx*) * [(*gx***hx*)*(*fx***hx*)]}

LC would probably capture this hierarchical inference by means of a leveled type structure in the Catalogue. Again, there is more than one way, but I would suggest:

HIERARCHY ByTaxonomyAs

(All.*~<Kingdom.*~<Phylum.*~<Class.*~<Order.*~<Genus.*~<Species.*

While no less complicated than the Predlog expression (in fact more so, since I’m including distinctions not necessary to the inference), the LC hierarchy of 1-n relations is fare more intuitive than the former with its nested if-thens. And so, assuming an instance ‘Whale’ (or ‘Cetacea’) under the type ‘Order,’ connected to ‘Mammal’ under ‘Class,’

[n] The ‘or’ of (*x*) (*gx* v *hx*) * *fx* indicates a ‘union’ of properties, which implies a move from the sets *gx*, *hz* (here, ‘rectangle’ ; ‘rhombus’) to the *superset fx* (‘four-sided polygon’). The ‘and’ of (*x*) (*gx* & *hx*) * *kx* indicates an ‘intersection’ of properties, and thus a move from *gx*, *hx* to a proper *subset kx* (‘square’).

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

connected to ‘Vertebrate’ under ‘Phylum’ (that is, assuming background schemata of the form ‘Phylum, class.mamal.at_below = vertebrate’ : and, ‘Class, order.whale.at_below = mammal’) then a query of the form ‘Whale.*~Phylum should yield the correct instances-match vertebrate – and likewise for any set-superset inferences we might wish to query of our Taxonomy.

In general, syllogisms and the like in LC defer to the question ‘What was the background schema/typestructure?’, since these function as the template for aggregating and truth-testing. Of course in some ways this misses the point: while we might argue that similar type structures mediate all our reasoning – say, ‘If Socrates is a man, and all men are mortal, then Socrates is a mortal’ – referencing those schemas has nothing to do with the *truth* of the syllogism. As with all the Prop-and Predlog inferences we’ve seen, it is true by form, that is, regardless of the truth values and aggregations of the individual arguments. It would have been just as valid (i.e., true) to infer, ‘ If Socrates ia a Martian, and all Martians are twelve-toed, then Socrates is twelve-toed.’ Similarly, if the types are structured in a certain (logical) manner, they will function inferentially, i.e., as true-by-form. Whether these inferencings tell us anything about the world depends on the integrity of the data source. But if the data is sound, then LC has a cleaner and more direct approach to these forms of hierarchical reasoning than Predlog, or for that matter, set theory.

While, in one sense, hierarchy could be considered the principle ordering feature of Predlog and Set theory (reading ‘*fx*’ as ‘*x*f*),[m] many posits of a many-to-one aggregation do not a hierarchy make. It could be argued, in fact, that the primitive

[m] Say, ‘the house is red’ = ‘The house belongs to the class of red things.’

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

(undefined) nature of the epsilon-relation is what impedes the structuring of type hierarchies.

First of all, *fx* (or *x*f*) isn't *transitive*: to be a member of a set isn't, by virtue of that fact alone, to be a member of a set of that set. Which is correct enough, since transitivity, characterizing part-whole relations, is just a special case of the various uses we ascribe to the functional/epsilon notation. But more correctly – whether or not it is well understood – this resistance to multiple aggregation stems from the notation itself: in *TLP*-terms, *fx* is a *function*, not an *operation*, and as such Q.) it belongs to the assertion-schemata of LC (as opposed to logical operations, which belong to the command/query syntax, within which types are structured); b.) it specifies the prototype of its argument – *f* of *x* – and so cannot sensefully take itself as its own argument (as opposed to operations, whose results are functions, and whose results can be taken as their own base; and so c.) it is not recursive (as opposed to logical operations – such as 'N(f)' or T.*>~T.* -- which are recursible, and can be applied over multiple iterations.

Of course these three all say the same thing, really, which is really the grounding insight here: the prototype specification intrinsic to the *f* of *x* is *Content of a location*: or call it what you will – but we still have to mention and treat *f* and *x* distinctly, [m] and we can't assert a content of a content, only of a location[n]:

---

[m] *TLP* 5.5261. "They both, independently, stand in signifying relations to the world."

[n] *TLP* 3.333. The notion that a function could be its own argument derived from class theory. Since certain classes seem to be members of themselves – the class of all classes, for eg. – we should be able to symbolize this – *fx* * *fx* – and so likewise its complement -- ~(*fx***fx*) – for all the ones that are not self-membered. Then of course we could define and reason about the class F of all the non-self-membered classes: for any class *fx*, *fx* * F iff~ (*fx***fx*). And of couse we would want to know of this class if *it* was self-membered or not – which by plugging F in as a value of its definition yields F* F iff~(F*F); i.e. Russell's class paradox: conceptually at least, the mother of all paradoxes. *TLP* 3.333 disarms it at the first move (using the functional notation, following Russell's "no class" translation of epsilons into propfuncs). "The reason why a function cannot be its own argument is that the sign for a function already contains the prototype of its argument, and it cannot contain itself. For let us suppose that the function *F*(*fx*) could be its

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

- And so functions (propositional, truth-) of the assertion-schemata while not themselves recursible, are the results of recursive operations (or refer to type structures in the catalogue that are).
- And furthermore, the operations that structure hierarchies implicate *an extension of logic itself*. That is, just as Predlog extended the purview of propositional logic by bringing into formal consideration distinctions that were only implicit to *p*'s and *q*, so the orderings structured in LC – between types, and between the instances of a type – are equally extensions to what

---

own argument: in that case there would be a proposition '*F*(*F*(*f*x)),' in which the outer function F and the inner function F must have different meanings, wince the inner one has the form * (*fx*) and the outer one has the form *(*(*fx*). Only the letter 'F' is common to the two functions, but the letter by itself signifies nothing.

The point is that on a functional interpretation of the logical variables (meaning here a *usage-based* one) 'F' isn't a class name for 'predicates' or 'classes' or even (or especially) 'contents'; it signifies a usage of its values, for which it stands proxy. That is why in LC syntax there is not particular need for an 'F' or 'C' token, since the form of a query *shows* which types function as locator and which as Content. 'F,' 'x,' etc. are given in this paper, since we need to show how Predlog translates to LC-log (and hence, LC), and in its own terms; and because in this context we are forming inference schemata outside of a specific query-context that would show the prototype distinction.

L.W.'s point, then, is that, functionally speaking, we cannot meaningfully symbolize or assert that a class is a member of itself (from which it follows that we can't assert its negation, that a class isn't a member of itself.) Only on a referential reading of the variables would we be tempted to call the inner 'F' of an '*F*(*F*(*x*)' a 'content' of that propositional sign ("because F stands for content"). But the outer 'F' shows that the inner one can't functionally be playing that role.

Of course I'm ignoring the other half of this argument, the notion that *f, x, n* (number), fx (function/class) f (x,y) (function/relation), *p* (proposition), etc, are *formal concepts*, and unlike concepts proper ('lion,' 'to the left of,' etc.) are symbolized by variables, which are not names for their domains, but formal usages regulating over discourse. So the formal concept *p* is not the class of all propositions, but the logical form that allows us to form all possible propositions (as in the Kantian transcendental object *x*, which allows us to conceive all possible objects (i.e. *as objects*); except that form L.W. these distinctions are drawn linguistically rather than epistemologically.) And so the formal concepts themselves (as symbolized by the variables, eg) are not quantified over; their values are (In LC terms, the prototypes are not a 'Type of Types,' i.e. they are not related to the types by degree of generality, but by a usage of them). And so, just as the formal concept '*n*,' 'number,' is not *itself* a number – or even the number of all numbers – or 'proposition' a proposition, the "class" of all classes is not itself a class (clearly, since we can form the class of that class and all its classes – or the power set of all its subsets, etc – ad infinitum); an so isn't a member of itself (Understand, I'm talking about the prototype here, the class concept itself, not the largest set of a database, even of 'the class of all classes extant in the universe at this moment.' So Russell et al. would not accept a time-based circumvention of the atinomy, since nothing could prevent us from querying about 'all classes for all time instances.') Between this notion and the former (that we cannot symbolize a function taking itself as argument – or in LC – speak, a content of its own content – since a content is by form always of a location Wittgenstein can conclude 3.333, "That disposes of Russell's paradox." [For a fuller treatment of this, see Appendix II from 1998].

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

can be considered properly logical inferencings. (while presently such will be beyond the restricted scope of this paper to prove yet from our earlier use operations to derive both Prop- and Predlog, it can be seen how this is doable.)

- And finally, hierarchies structured as the iterations of a recursive operation are internally related, and thereby formal transitivities.

The school of logic most concerned with transitive part-whole relations is mereology (or mereotopology), which traces to Leonard and Goodman (1940), and more recently Lewis *Parts of Classes*.

In mereology, first, the members of sets are *defined* as sets themselves (with the singleton, or unit-set given as the primitive element, and second, transitivity is postulated as an axiom: $(x)(y)(z)[(Rxy\&Rxz)*Rxz]$. Of course this stipulation is not of itself a tautology, and is just that, a stipulation. Moreover, it is not a generally recursive form (i.e., an operation), and, from an 1-c perspective, only expresses one sort of mereological aggregation: i.e. a locator/dimensional hierarchy in contradistinction to a content-based hierarchy. Just as we saw earlier how different inferences can be made from relations between Content Types and from relations between Locator types, so different aggregation pathways can be structured as well. (In fact, the one follows from the other.) And so the axiom form given above is to be strictly distinguished from a content-based aggregation, eg. $(x)[(fx*gx)\&(gx*hx)] *(fx*hx)$. This, at least, is prima facie in modus ponens – i.e., *is* a tautology; but again, it is not in a generally recursive form. (This is, of course, only a detraction once we've formalized the operation from which it – and similar functions – derive.) The point of recursiveness is not transitivity *per se*, but iterable resolution: we want to be able to say that 'city' is a political sub-unit of 'county' in the

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

same or similar-enough sense that ‘county’ is a sub-unit of ‘state,’ and ‘state’ of ‘country’ (within which relations, the transitive ones follow *tout court*).

*LC Set Theory and RM*

The reduced focus of the present paper will not let to bring full blown set theory into the mix, but I have a few off-hand remarks to make on the subject, since it closely relates to LC, and to the algebra for RM. On the one hand, the algebra for classes is elegant in its simplicity, and far cleaner and easier to work with than Predlog. However, the * relation of member to set conflates two very different sorts of relations in LC:

1.) First (and correctly foremost), the relation of L to C, or $x$ to $f$ in Predlog.

2.) The relation of instance to type, or value to variable in Predlog.

Now it is true of course that the relation of type to instance *is* a set-to-element correlation: in the type structures of the catalogue. (In this sense the type structures define a set of proto-propositions: ‘red is a color,’ etc.) But in LC there is good reason for defining propositions-in-use as the correlation of two type-instances. And this approach conforms precisely with 1.), the primary use of * in set theory, as a straightforward translation of the propositional functions of Predlog. So, referring to our A,E,I,O prepositional forms, the propfunc notation can be reformulated in terms of the three main notations of set theory:

$$\underline{(x),*} \qquad \underline{=} \qquad \underline{*}$$

$$\mathrm{A} = (x)(x*A*x*B) \equiv \ \bar{A} + B = 1 \ \equiv \ A*B$$

$$\mathrm{E} = (x)(x*A*x*B) \equiv \ \bar{A} + B = 1 \ \equiv \ A*B$$

$$\mathrm{I} \ = \sim(x)(x*A*X*B) \equiv AB \neq 0 \qquad \equiv \sim(A*B)$$

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

$O = \sim(x)(x*A*x*B) \equiv AB \neq 0 \quad \equiv \sim(A*B)$[n]

*A,B* = sets/classes/(Types)

*a, b* = members/elements/(instances)

* = 'member/element/(instance) of'

1 = universal set/class

0 = null set/class

c = 'included in'

$\bar{A}$ = '*x* is not a member of A (set negation/class complement)

t = 'or' (set/class sum)

AB = 'and' (set/class product, i.e. 'x')

If it isn't already apparent, the equivalence I'm aiming at here is 'A*B[and its equivalencies] ≡ T>~T′.' Bearing this in mind, we can now combine LC-log with the concept of *functions on a set* for the purposes of formally correlating (i.e. reducing) RM to LC. In RM, as we've seen, the *f* is the relation *R*, symbolically external to the attributes/columns: eg. The assertion *f*(*x,y*) is a relation in which *x* ranges over the domain of set *A* (say, 'Stores') and *y* ranges over the domain of set *B* (say, 'Sales'); and so *A* x *B* is the set of all values for *x* and *y* assertible in *f*(*x,y*). Of course (and leaving aside for now the various ambiguities associated with the *RM* notion of 'domain' or 'type'), the immediate difficulty is that not all elements in the domain of an attribute are asserted in a relation. The relation itself is usually a subset of '*A* x *B*' (otherwise the RM concept of 'relational complement' – the tuples in the domain of *A* x *B not* asserted in some relation

[n] Modified from Ambrose/Lazerowiz, 64.

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

of the form $R(A,B)$ – would have no useful meaning.[n] The relation itself is a proper subset of AxB, called in set theory the *truth-set* – i.e., the values of $x$ and $y$ that make $f(x,y)$ true. The truth set, then, *is* the relation, which can be defined as,

$$R = f(x,y) = \{(a,b) * A\text{x}B : f(a,b)\}$$ [m]

In other words, a relation from $A$ to $B$ is equivalent to a function from $A$ to $B$, which is expressed as a proper subset of $A$x$B$.

So far we have used the function $f$ in LC-log as propfuncs – functions that map values onto propositions. Similarly (operationally), we can think of these (scalar) functions as mapping the elements (instances) of one set (type) onto elements of another, or $f : A \rightarrow B$ in functional notation.

Def. 1.) $f : A \rightarrow B$ iff $(a)*A(*\text{b})*\text{B}[(a,b)*f]$

We can go to co-specify the domain and range of the function/relation as

$$\text{Dom}(f) = \{a*A: (*b)*B[(a,b)*f]\}$$

$$\text{Ran}(f) = \{b*B: (*a)*A[(a,b*f]\}$$ [n]

By Def.1.) *every* element of $A$ must appear as the first coordinate of some (i.e., exactly one) ordered pair in $f$. Therefore the domain of $f$ is *all* of $A$ – which is precisely what is specified in the LC syntax by the dot-star notation: eg. $f : A \rightarrow B$ = Store.*~Sales. Moreover it is clear by the definition the range of $f$ is need not be all elements of $B$, but

---

[n] Date/Darwin: "The body B*r* or [relation] *r* is a set of such tuples *tr*. Note that (in general) there will be some such tuples *tr* that conform to [the heading] H*r* but do not appear in B*r* "(p. 54)." The NOT operator yields the complements *s* of a given relation *r.* The heading of *s* is the heading of *r.* The body of *s* contains every tuple with that heading that is not in the body of *r*" (p.55). "More precisely, *any* relation can be regarded as an operator that maps from some subset of its attributes to the rest …" (48).

[m] In traditional relation theory $f(a,b)$ is an *ordered pair*, i.e., a relation of $A$ to $B$, and not vice versa ($a$ is to the left of $b$ does not say the same thing as $b$ is to the left of $a$). For Codd, however, it was important that the order of attributes in a relation be of no consequence – which of course has consequences. Wiener (1914) reduced the logic of relations of that of classes by treating the dyadic relation $x$R$y$ as an ordered pair $(x;y)$ in set theory. Quine later (1945/6) extended this to sequences of finite length $(x;y;z;w\ldots)$.

[n] Major assist for this and much of the following goes to Daniel J. Velleman, *How to Prove It: A Structured Approach*.

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

only those instances that appear as the second coordinate of the ordered pairs in *f*.[m] In the set-function terminology the second coordinate in f is called the *image* of the first; the range of *f* can therefore be redefined as "the set of all images of instances of *A* under *f*, or

Ran(*f*) = {*fa* : a*A}[n]

It should be clear that the same goes for ordered triples or *n*-tuples – and furthermore, that *a function discrimination* of those tuple values that specify domains from those that specify range *is precisely that discrimination of dimensions* (or "dimensional coordinates") *from variables* : *that is to say, locators from contents*. It is likewise the key for logically translating (reducing) RM to LC. But I'll first need to address a logical – or I hope merely terminological-snag.

The LC query syntax uses the set-theory (ST) tilde for relating types, say, 'Stores.*~Sales,' or "Store-dot-star one-to-one-Sales." Actually in ST the tilde means "equinumerous with" as in "the set of Stores is equinumerous with the set of Sales figures." And in ST two sets A and B are defined as equinumerous if there is a function mapping A to B that is both *one-to-one* and *onto*. Now *one-to-one* is defined as (reverting here to the logical tilde, for 'not'):

$\sim[(*a_1)*A(*a_2)*A(fa_1=fa_2 \& a_1 \neq a_2)]$,

which is fancy for "It is *not* the case that there are two distinct elements of *A,* $a_1$ and $a_2$, that map onto the same value" (i.e. in set *B*; in other words, that $fa_1$ cannot have the same value as $fa_2$. A mapping is *onto* iff:

$(b)*B(*a)*A\ (fa=b)$,

[m] So that f(a,b)*AxB, the truth set in RM, equates to what I would call the *assertion set* in LC. See below where I discuss 'one-one' and 'onto.'
[n] Read as 'the range of f is the set *f* of *a* such that *a* is an instance/element of *A*.'

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

that is, every element of *B* is a mapping from some (at least one) element of A. Now taking 'onto' first: whether this condition holds for LC depends on what you consider to be the actual instances of Type/set *B*, (again, say, Sales) whether a) all possible Sales values (i.e., the specific unit layer, or: Sales.units.*); or b.) all actual sales values that appear in any context (so, Sales, Sales.* might list reported sales values from 'Dept.' and 'Region' as well as 'Store') or c.) only the actual sales values for Stores.*. Option a.) I would call the 'assertability set,' belonging more to the 'logical space' of possible assertions about sales. Option b.) strikes me as the likeliest candidate for actual instances of the Type 'Sales' – that is, if types are to have some sort of independence in the catalogue (that is, outside of some usage).

Option c.) then represents a subset of AxB (because it is a subset of B, vis-à-vis options b.) and a.)), and corresponds to the 'truth-set' definition given above.[n] To give another example: in 'Object_Shape.*~Color' it may happen to be that not all instances of Type 'Color' are asserted in the schema (it may happen that there is not object instance that is 'red'). However, when not all instances of a content-type (set *B*) are mapped onto, the function/relation is not *onto*, and by definition not 'equinumerous' (~).

Likewise, with regard to *one-to-one* – the question again turns on what is to be regarded as the actual instances of a type. Let's say, for eg., that the function 'Object_Shape.*~Color *is* onto: even still, if more than one instance of set *A* – say objects

[n] Although in LC terms, I would prefer 'assertion-set' – i.e. the subset of instances of B that are *asserted* to be true of their corresponding instances of A – but may not be, since every senseful atomic prop is true or false (i.e. asserted to be true, but if so, still contingently so). RM of course following PM, *postulates* that the set of atomic props are *true*. The point is minor, yet it has consequences, for eg., when tuple values actually prove to be false and require updating. Conceptually (and physically) the replacement of a tuple value in RM is equivalent to replacing the entire relation (although RM does this effectively enough that the user is effectively unaware of this fact: in fact, RM may be *too* effective at this type of update – considering how the Diebold electronic voting machines have proved to be "updatable" after-the-vote, and without leaving any electronic trace of it).

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

'car' and 'barn' – are assigned the same color instance – say 'green' – the relation is *not* one-to-one, and thereby not 'equinumerous' (~). For example:

| Type 'Object' | Type Color | Type 'Object' | Type 'Color' |
|---|---|---|---|
| car | red | car | green |
| barn | green | barn | blue |
| house | blue | house | blue |
| camel | magenta | camel | yellow |
| | yellow | | |
| 'one-to-one,' but not 'onto' | | 'onto,' but not 'one-to-one' | |

The same goes, of course, for two stores that have identical sales values, say '150.' (To obviate this you would have to claim that this '150' and that (or this 'green' and that) are not the same instances of the Type, although I'm not sure how, without that is, undermining the logical independence of the Types.) The only way around the quandary here – at leas that I can see – is to qualify the sense of '~,' for eg., by considering only the mappings themselves as ~: in the parochial sense that a tuple value is mapped to a tuple value, and not drawing any conclusions about the set-to-set relations between types. (One option is to fall back on L to C *predication* as the basic (primitive) relation here: but then I might be missing something – I really only have the DDL doc to go on…)

*Reducing RM to LC*

Returning to our definition of the RM *truth set* (which is, from the foregoing, equivalent to the *assertion-set* of an LC function):

f: A→B = {(*a,b*)*$A$x$B$: f(*a,b*)}

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

In other words, for *a*A*, there is some value *b*B* such that (*a,b*)***f*. And given this set, we can ascertain which ordered pairs are elements of *f* by specifying for all *a*A* and *b*B* (*a,b*)***f* iff *b* = *fa*, i.e.:

*f* = {(*a,b*)*AxB: b=fa}

In other words, we have moved from an RM expression, where *a* and *b* are both *arguments* (locators) of an external dyadic predicate (content), to one in which the *f* can *substitute for b*: *b* is 'the "image" of *a* under *f*,' or 'the value of *f* at *a*, or just *f* of *a*. The variable *b*, as a co-argument at *a*, can be replaced by *fa*, in which the * is replaced by predication (i.e., the "operational" function of mapping from set to set becomes the L-to-C propositional function), and the set/type B is represented by its proper subset, the variable *f*.

So, eg., if (*a,b*) * *A*x*B*: *b* = *a*'s Sales in dollars, then

(Store*, 150) *f [in RM]

≡ f(Stored) = 150 [in LC]

The LC variable *f* specifies a proper subset of B, one of whose values/instances is '150.' It is a reduction because the extraneous Relational predicate – which does nothing except assert that a relation holds between the arguments (or by its heading-name, helps the user understand what-the-hell is being asserted about those arguments) – now plays a logical role in the assertion. For example R(*x,y,z,w*) – say, R (Store, time, product), in which Sales is treated as an instance of the variable *f* at each cross product for the instance-coordinates *x,y,z*.

In conclusion: the formalization of LC/LC-log in terms of Prop- and Predlog has given us a rather lean and elegant set of primitives for treating all the various truth-

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

functional relationships expressible in canonical logic. Of course these are only a subset of the orderings expressible in full-blown LC – but those are beyond the scope of the present paper. However, given the foregoing, I consider it more than interesting to find that Date and Darwin in the *Third Manifesto* of RM come down to a similar conceptual simplification of their Relational Algebra (A):

[T]hus we could in fact reduce our algebra still further to just the two operators REMOVE and either NOR or NAND (plus the TCLOSE operator once again, which is still to be discussed. (p.51).

- REMOVE "is the A counterpoint to the existential quantifier of predicate logic. It corresponds to Codd's *project*" (p.49).

- NOR: "So we could dispense with AND "and" collapse NOT and OR into a single operator NOR ('neither *A* and *B*'; equivalently 'not *A* and not *B*') "(p.50).[n]

NAND of course is the composite operator NOT-AND (='not *A* or not *B*'), which by the De Morgan laws (and the original Scheffer-stroke, '*p*|*q*') is equally effective and "relationally

complete."

- TCLOSE is not past of the reduction, but rather an addition to Codd's original algebra: it is the "transitive closure" operator, which live the transitivity axiom of mereology attempts to capture

hierarchies in RM.

[n] Notice the counterintuitive result that N(*), the set-negation operator which negates all values, p *and* q, etc., of * (yielding '~p&~q,etc.') is not decomposable into the assertion relations '~" (not) and '&' (and). It is if anything – i.e. if not treated as primitive – a composition of the operators NOT and OR: this seen readily from the truth tables, from the fact that the double negation NN(*) yields '*pvq*, etc.'

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

NOR, of course, is ovr N(*). The choice of the existential quantifier over the universal is interesting. Either quantifier can be specified entirely in terms of the other, as we have seen. I have been giving preference to the universal, first because it reads more universally – 'all' is easier to generalize on a variable than 'some (at least one),' and second, because of the prevalence of dot-star schemas in LC. But in some ways (*$x$) relates more immediately to N(*), since NN(*) = pvqvr….=(*x)fx. Finally I'd note that as TCLOSE extends Codd's algebra, the recursible operation T.*~<T′.* similarly extends logic by generating truth-functional orderings of propositions not easily captured in the Prop- or Predlog specifications.

This last part, while beyond the scope of relating LC to Predlog, hinges on certain points that have been made here, namely the truth-functional (extensional) equivalency of Proplog and Predlog, and the generation of both by means of an operation: in fact, the same one, the only difference between Prop- and Predlog how one specifies the values of the variable *. So, Wittgerstein writes:

We *can* distinguish three kinds of descriptions: 1. direct enumeration, in which case we can simply substitute for the variable the constants that are its values; 2. giving a function *fx* whose values for all values of *x* are the propositions to be described; 3. giving a formal law that governs the construction of the propositions, in which case the bracketed expression [(*)] has as its members all the terms of a series of forms. (*TLP* 5.501)

Just as the first two of these correspond to Proplog and Predlog respectively, so, I would argue, the third corresponds to the *internal* relations generated by the direct

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved

ordering of Type structures in LC.[m] And as 1. is traditionally termed "specifying the variable by *extension*," and 2. traditionally as "specifying by *intension*," so, I would suggest, we call the (background, i.e. operational) additions/schemas LC brings to the table "specifying by means of *subtension*."[n]

[m] "I call a series that is ordered by an *internal* relation a series of forms. The order of the number-series is not governed by an external relation but by an internal relation … (If *b* stands in one of these relations to *a*, I call *b* a successor of *a*.)" (*TLP* 4.1254).

[n] From L. *Subtendere*, 'to stretch beneath' (undergird), as in def. 2 for 'subtend': 2a.)" To underlie so as to include. b.) to occupy an adjacent and usv. Lower position to and often so as to embrace or enclose." *Websters*, 10th ed.

 copyright Shavel/Thomsen © 2008 DSSlab al rights reserved