1 Introduction

This appendix breifly describes the benefits and omissions of Canonical Logic which consists of the Propositional Logic and the Predicate Calculus. It also shows how the LC Model directly supports Canonical Logic while providing principled solutions to some of its omissions. In this sense the LC Model extend Canonical Logic. A more detailed and formal treatment of these topics can be found in appendices (...)

Sections 2 and 3 describe the benefits and omissions of canonical logic. Section 4 describes how the located content model has equal representative power to any canonical logic system. Section 4.3 describes how the located content model settles some of the omissions in predicate logic.

2 Canonical Logic

2.1 Propositional Logic

Traditional propositional logic consists of propositions (e.g. p, q), relations between them: conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$) and negation ($\neg$) and inferencing rules which allow conclusions to be drawn from a collection of statements. Propositional logic is at the heart of rule-based systems in relational databases. An event trigger that generates an award when a certain amount of foodcakes are sold is nothing more than the propositional implication p $\rightarrow$ q, where p is the event "sales $>$ x" and q is the award.

2.2 Predicate Calculus

Whereas the propositional logic only represents the truth or falsity of a proposition, the Predicate Calculus represents the components of the propositions (i.e., the subjects and predicates, hence the term "Predicate Calculus"), By representing the components of a proposition, the Predicate Calculus enables statements to be made about specific aspects of the world. For example, the Predicate Calculus can make statements which are true of everything (universal quantification: $\forall$) and statements which are true of at least one thing (existential quantification: $\exists$). Notationally this introduces a *predicate* or statement which when applied to an object or instance becomes a valid proposition.

2.3 Truth Maintenance

The primary use of the predicate calculus, indeed its reason for being, is the application of inference rules to draw logical conclusions from assertions. In any information system this translates into truth maintenance, or the automatic extrapolation of logical conclusions from stored data. To use a traditional, if somewhat simple, example, a user may enter the following information into a predicate logic system:

- $\forall x$ Man(x) $\rightarrow$ Mortal(x) : "This reads All men are mortal"
- Man(Socrates) : "Socrates is a man"

From this information, our system, in the cause of truth maintenance can easily conclude Mortal(Socrates) or that Socrates is mortal.

In a more relevant example, we may assert

- $\forall$ i $\forall$ p (Polluted(i) $\wedge$ IsIn(i,p)) $\rightarrow$ Polluted(p) : "If an ingredient is polluted and in a product, then that product is polluted"
- IsIn(Cod, Fishcake) : "Cod is in Fishcakes"

This sets up the rules of our system. At some later point the following is entered:

- Polluted(Cod) : "Cod is polluted"

From this the system can conclude that our fishcakes are polluted. Of course this assertion may result in further inference which indicates that polluted fishcakes should not be sold; that polluted inventory is removed from stores; suppliers of polluted ingredients should not be paid, etc. All of these logical conclusions cascade from the single assertion "Polluted(Cod)."

Another important part of truth maintenance is the fact that the system can detect when there are inconsistencies in the system. Consider a system with the three previous statements already in place. If a statement such as:

- $\neg$ Polluted(Fishcake)

is entered into the system, the logical inconsistency is automatically detected. The system knows one of its four statements must be incorrect and it can prompt the user for resolution.

The rules of inference are cleanly defined and insures that all possible logical conclusions are immediately drawn and any logical inconsistencies detected. The ability to automate maintenance is the primary use of canonical logic in computerized information systems.

2.4 Generality

Canonical logic is of general use because its simple rules are applicable to all situations. They are not rules defined for specific environments or situations. Modus ponens doesn't care if a proposition is concerned with astronauts or astroturf. The logical conclusions that can be drawn from a set of assertions are independent of any "real-world" meaning that we may attach to them. To the system's point of view they are merely functions and symbols - little tags of data being pushed from place to place.

3 Omissions in Canonical Logic

The difficulties with Canonical Logic are not difficulties derived from Canonical Logic getting it wrong. Instead they are derived from what it leaves out. The omissions are examined below.

3.1 Propositional Logic

If we construe propositional logic as a mathematical environment of inferencing rules over propositions which are guaranteed to be valid (e.g. true or false), then propositional logic is a closed system, fully functional without omissions. The difficulty is in the admission into this closed system of propositions which are invalid for various reasons as detailed below in section 3.2.

3.2 Omissions in Predicate Logic

In moving from propositions (which must be, by definition, true or false) to predicates expressed about objects, we admit the possibility of meaningless or missing statements, causing a breach in completeness of our system.

3.2.1 Implicit Schemas

In predicate logic, there is no ordering relation between objects over which predicates are defined. Imagine we would like to make the assertion "if sales for this month are less than the sales for last month assert a warning." In predicate logic a "month" is simply an object, there is no notion of previous or next, establishing such a relationship would require separate predicates, for example: "Previous(m,n)" or "Next(m,n)" moreover, to insure integrity we would also have to add the rule "Previous(m,n) $\leftrightarrow$ Next(n,m)."

Predicate calculus does not exist in a multi-dimensional space, however its use in relational databases implicitly builds a multi-dimensional model. A location in this multi-dimensional space is examined by using predicates to fix values in each of a variety of dimensions. This means that any rigor or precision in the specification of this dimensionality is outside of the system rather than built-in.

3.2.2 Integrated Mathematics

The other weakness in the predicate calculus is the absence of traditional mathematics. We may introduce predicates "GreaterThan" or "LessThan" but they are no different than other predicates, unless we assert rules such as the canonical transitivity of the greater-than operator (If x is greater than y and y is greater than z, then x is greater than z)

- GreaterThan(x,y), GreaterThan(y,z) $\rightarrow$ GreaterThan(x,z)

Such notions are related to the lack of ordering relationships.

An additional difficulty is in expressing the notion: "The total sales for a store are the sum of the sales for each department." Without the integration of mathematics such a statement while quite useful is impossible.

3.2.3 Type Theory

A large omission in the predicate calculus is a Theory of Types that accounts for the data upon which assertions and operations can be made. Our earlier statements regarding Socrates' mortality implicitly define a a function, Mortal, which is applicable to things of type Man. The predicate calculus, however, is essentially an untyped programming language this means that there is nothing in the language which insures that a predicate we assert about a value has any relevance on that value. To give a concrete example, there is nothing in the calculus which prevents an assertion such as:

- $\neg$ Mortal(Mt. Everest)

despite the fact that our Mortal predicate is clearly ill-defined when applied to mountains.

Of additional concern is the fact that our quantification, universal or existential occurs over everything, regardless of type. Thus if we want to say something is true of all men, we must first universally quantify, and then restrict with an implication. We might wish to say that all fishcakes contain fish. We may hope that we could just say $\forall$ fishcake, IsIn(Fish, fishcake), rather we need to say: $\forall$ x Fishcake(x) $\rightarrow$ IsIn(Fish,x). The integration of a type theory allows us to make statements not over all objects but solely over a particular type.

Another omission in the Predicate Calculus that stems from its lack of Type theory is the notion of hierarchies within and between types. For example if it is prohibited to go anywhere in Europe this should imply that it is prohibited to go to Italy. In canonical logic there is no representation of this containment relationship, it must be explicitly specified by the system architect.

It is worth noting that a crude type system could be constructed from predicate logic, much as set theory and mathematics were derived. But the Predicate Calculus provides no support for any such Type system. And further, the primitive requirements for any viable Type theory include notions of adjacency and function that are simply not a part of any aspect of Canonical Logic.

3.2.4 Invalid and Meaningless Expressions

Finally, the predicate calculus lacks a systematic way of dealing with invalid, meaningless or missing expressions. There are several layers of meaningfulness/ meaninglessness as relates to queries and assertions

Predicate calculus takes care of two of them:

Undefined predicate You can not assert a predicate which is undefined.

Undefined object You can not assert something about an object which is unknown.

However it leaves three undefined:

Invalid Application Applying a predicate to an object for which it is ill-defined, e.g. IsIn($100, Fishcake)

Missing Data LessThan(Sales(Region), $100), when the sales value is missing for a particular region.

Under-Specification A location of unique content may require specification by three dimensions, what happens when predicates are only used to fix two of them?

Under-Determination IsSource(Fish, Cape Cod), while it is true that Cape Cod is the source of some of the fish, it is not the source of all of the fish.

4 Relating the LC Model and Canonical Logic

This section shows how the propositional and predicate calculus can be represented in the LC Model using the concrete syntax proposed in sections 5 and 6 of the LC Model document. Please note the following.

1. There are two distinct levels of representation, both of which are described. First, there is the meta-representation. This is the description of what is common to all propositional logic and predicate calculus schemas. Then there are specific examples of how LC is used to do the same specific thing that Canonical Logic might be used to do.

2. The Type and value names in LC were chosen to highlight the mapping form Canonical Logic to LC. As such, they use considerably more ink than the native tokens in Canonical Logic. This in no way implies that LC is more verbose than Canonical Logic.

4.1 Propositional Logic

The following expressions represent the general schema for propositional logic.

```
create type TruthValue with units Boolean
create type Proposition with units Characters

Proposition.*[1]-[1]+TruthValue
```

Each proposition specifies a location and its content, for example som proposition "p" might denote "`Location.US,Store.423`"

To assert a proposition, p:

```
TruthValue, Proposition.p = true
```

To negate a proposition: $\neg p$:

```
TruthValue, Proposition.q = false
```

To assert the conjunction of two propositions, $p \wedge q$:

```
TruthValue, Proposition.p = true AND TruthValue, Proposition.q = true
```

To asert the disjunction of to propositions, $p \vee q$:

```
TruthValue, Proposition.p = true OR TruthValue, Proposition.q = true
```

To assert the implication of two propositions, $p \rightarrow q$:

```
IF TruthValue.true, Proposition.p THEN TruthValue.true, Proposition.q
```

4.2 Predicate Logic

The following expressions represent the general schema for the Predicate Calculus:

```
create type PredicateFunction with units Char
```

Note: this treats all predicate calculus predicates as values of a single Type called "PredicateFunction" Though it might ofend our sense of semantics to think of "Mortal" "Green" "$100" and "noisy" as vales of the same Type, keep in mind that the Predicate Calculus has no notion of Types, just predicate symbols and argument symbols.

```
create type
  Argument with units Characters AS ObjectClass and ObjectInstance
    with Ordering Relationship ObjectClass.*[1]-[N]ObjectInstance.*

create type TruthValue with units Boolean  and Values {True, False}
(PredicateFunction[1]+,Argument[1]+).*[1]-[1]+TruthValue
```

This reads, "Every unique predicate function and argument has a truth value"

Given this, and given the assertion that "Man" is a value of Type "Argument" with units "ObjectClass", our earlier predicate example about Socrates can be expressed as:

```
IF TruthValue,(Argument.objectInstance = Man.down(1)) = True
   And TruthValue,(PredicateFunction.Mortal, Argument.Man.down(1).*)
     = True
THEN
   TruthValue (PredicateFunction.Man, Argument.ObjectInstance.Socrates)
     = True
```

From these assertions and following the rules of inference, the LC system can draw the same conclusions. A full suite of inferencing examples are shown in the appendix to this section.

4.3 Addressing omissions in canonical logic

The primary addition which the LC model brings as an addition to the predicate calculus follows from its theory of Types, including its definition of well formed schemas. The LC Model addresses the omissions in predicate calculus mentioned earlier.

Since in LC the primary designator of structure is the schema of data, the schema behind the data can, and should be explicitly specified.

4.3.1 Ordering schemas and dimensionality

In LC, the ordering relationships in a type are explicitly defined. Thus our earlier example "If this month's sales are less than last month's sales trigger a warning" is easily expressed in LC as: `If Sales, Month.this < Sales, Month.prev then Warning=true`

Furthermore the specification of schema for our data means that the dimensionality of any particular location is concretely specified and the manner of specifying values in each dimension are represented as part of the type rather than ad-hoc predicates. To make a statement about all stores in the US with fishcake sales greater than 100 units in predicate logic we need to fix the location with predicates: $\forall$ x $\forall$ y IsStore(x) $\wedge$ Location(x, US) $\wedge$ IsStoreSales(x,y) $\wedge$ GreaterThan(y, 100), and then continue on with our statement. In contrast in LC the specification of the schema insures that x is a store, and that y are the sales for that store, so we can simply say: `Geography.USA.Downlevel(Store).*, Sales > 100` to capture the set of all stores.

4.3.2 Integrated Mathematics

The LC model provides a unified treatment of mathematics and logic, so calculations and numeric comparisons are supported wherever the appropriate ordering relationships exist. In this way it is simple to specify things like "the sales of a store are the sum of the sales of the store's departments." In LC this is expressed as: `Sales, Store.* = Sum(Sales,Department.*)`

4.3.3 Typed Predicates

Another feature of the LC Type Model is that we can construct type operations instead of general, untyped, predicates. In contrast to the predicates concerning Fishcakes and pollution asserted earlier, we might instead say:

```
define type Polluted units Boolean
define type Ingredient units Character
define type Product units Character

Product.*[1]-[1]+Polluted
Ingredient.*[1]-[1]+Polluted
Product.*[1]-[n]Ingredient

if Polluted.true,Ingredient.* then Polluted.true,Product.*
```

Here we have asserted through our schema and a single statement, the logical conclusion we were attempting to draw earlier. Further the conclusion we have made is specific to Products.

4.3.4 Meaningless and Invalid data in LC

The LC syntax provides concrete decision classes for all possible categories of meaningless or invalid data as specified below.

Result of trying to execute the query	**Well constructed or not**	**Well formed or not**
1. The tokens do not correspond to any known Type names or values	Not a well constructed query	Not well formed
2. the tokens correspond to known Type names and values but the combination of Types referred to in the assertion do not correspond to any known Type Structure	Not a well constructed query	Not well formed
3. The tokens correspond to Type names and values in a known Type structure but they are all open or all closed	Not a well constructed query	Not well formed
4. the tokens correspond to a partially open (LC form-possessing) Type structure but in attempting to execute the query no matching location can be found	Yes a well constructed query is the same thing as "potentially meaningful" Not necessarily meaningful ;	not well formed in the classical sense
5. a matching location is found but more than one content is found	Yes a well constructed query	Yes potentially meaningful and probably not well formed in the classical sense but "under-differentiated" in LC
6. more than one matching location is found each with one or more contents	Yes a well constructed query	Yes potentially meaningful and probably not well formed in the classical sense but "under-differentiated" in LCese
7. a matching location can be found but no contents are found	Yes a well constructed query	Yes potentially meaningful and well formed in the classical sense but "missing" in LCese

A well formed assertion specifies which Type is asserted of which Type and requires minimum of three logical tokens: Locator value, content name, content value (or locator name locator value and content value assuming content values are globally unique).

Given a set of well formed Types such as Object name that includes "house"

as a value and Color that includes “Blue” as a value and given a partially open Type structure Object name .* Color

Given a well constructed assertion such as “Color of house is blue” and a sole data source as truth against which the assertion can be tested

The assertion is well constructed and true if the data source contains a single location called “house” with a single associated color value called “blue”

The assertion is well constructed and false if the data source contains a single location called “house” with a single associated color value called any valid color value other than “blue”

The assertion is missing/indeterminate if the data source contains a single location called house with a single associated possible color value that is actually missing

The assertion is meaningless (not necessarily meaningful i.e. potentially meaningful) if the data source contains no location called house

Potentially meaningful but under differentiated if the data source contains either multiple locations called house each with an associated possible color value or a single location called house with multiple associated potential color values

4.3.5 Other omissions

Canonical logic is also silent about a number of other topics that arise frequently in information modeling situations. These include inferencing across hierarchies and inferencing across time. Though beyond the scope of this section, these topics are systematically dealt with in the LC Model. Examples of time logic are included in sections ( ) . Examples of inferencing across hierarchies are included in sections ( ).

Examples The following appendix gives practical comparisons of business rules and ideas expressed in predicate logic and their corollary in LC. It consists of a section of functions (section .1) which are operations which return numeric values for use with predicates (section .2). The schema for use in LC is given in section .3. Finally a series of examples using these two environments is given in section .4.

.1 Numerical Functions

1. $Qty(d)$ The amount sold on day d
2. $Profit(d)$ The amount of profit for a given day d
3. $Inventory(d)$ The total number of items available for sale on day d
4. $Received(d)$ The total amount of products received on some day d
5. $Sales(p, d)$ The total sales on day d, given price p
6. $Sold(d, s, i)$ The total sales of item i at store s on day d

.2 Logical Predicates

1. $SoldMore(d, d')$ More was sold on day d, than on d'
2. $ProfitMore(d, d')$ The profit for a given day d was more than day d'
3. $Holiday(d)$ Day d is a holiday
4. $Delivered(t, d)$ Did truck t make a delivery on day d?
5. $Snow(w)$ is weather w snow?
6. $Drivable(w)$ can we drive in weather w?
7. $Weather(w, d)$ does day d have weather w?
8. $Location(s, c)$ Is store s in country c?
9. $Polluted(i)$ Item i is polluted.
10. $IsIn(i, p)$ Item i is in product p
11. $IsFiller(i)$ is item i filler?
12. $IsFish(i)$ is item i fish?

.3 LC Schema

```
create type Day with units Naturals
create type Sales with units Naturals
create type Profit with units Dollars
create type Holiday with units Boolean
create type Inventory with units Naturals
create type Received with units Naturals
create type Weather with units {Snow, Rain, Sun, Wind}
create type Drivable with units Boolean
create type TruckID with units Naturals
create type Delivered with units Naturals
create type Price with units Dollars
create type Product with units {Fish-cakes, Veggie-cakes}
create type Polluted with units Boolean
create type Ingredient with units {Cod, Shrimp, Flour, Carrots}
create type IngredientType with units {Fish, Filler, Vegetables}
create type QtyIngredient with units Naturals

   Day.*[1]-[1]+(Weather)

   Product.*[1]-[1]+(Polluted)
```

```
Ingredient.*[1]-[1]+(Polluted,IngredientType)

Weather.*[1]-[1]+(Drivable)

Truck.*[1]-[n](Delivered,Day)

(Product.* x Ingredient.*)[1]-[1]+QtyIngredient

(Product.* x Day.*)[1]-[1]+(Sales,Profit,Inventory,Received,Price)
```

.4 Statements

1. For any pair of days, selling more means making more

$$\forall d \forall d' (SoldMore(d, d')) \rightarrow ProfitMore(d, d')$$

```
if Sales, Day.v as x > Sales,Day.!v as y then Profit, Day.x >
Profit, Day.y
```

2. All holidays result in selling less than any non-holiday (and by modus-ponens making less too)

$$\forall d \forall d' (Holiday(d) \wedge \neg Holiday(d')) \rightarrow SoldMore(d, d')$$

```
Sales.max,Day.Holiday.1.* < Sales.min, Day.Holiday.0.*
```

3. Selling nothing means making nothing

$$\forall d (Qty(d) = 0) \rightarrow (Profit(d) = 0)$$

```
if Sales,Day.v as x = 0 then Profit,Day.x = 0
```

4. Some holidays result in selling nothing

$$\exists d (Holiday(d) \wedge (Qty(d) = 0)$$

```
Count(Day), (Sales, Day.Holiday.1.*) > 0
```

5. No inventory means no sales

$$\forall d (Inventory(d) = 0) \rightarrow (Qty(d) = 0)$$

```
if Inventory, Day.v as x = 0 then Sales, Day.x = 0
```

6. On some days receiving nothing means no inventory

$$\exists d (Received(d) = 0) \rightarrow (Inventory(d) = 0)$$

```
Count(Day), (Day,Received.0,Inventory.0) > 0
```

7. If nothing is delivered, nothing is received

$$\forall d(\forall t \neg Delivered(t,d)) \rightarrow (Received(d) = 0)$$

```
if (Delivered,Truck.t,Day.v as x) = 0 then
(Received,Day.x) = 0
```

8. Some snowy weather is undrivable

$$\exists w(Snow(w) \wedge \neg Drivable(w))$$

```
Count(Days), (Days,Weather.Snow,Drivable.0) > 0
```

9. All days with undrivable weather mean no deliveries

$$\forall d((\forall w Weather(w,d) \wedge \neg Drivable(w)) \rightarrow (\forall t \neg Delivered(t,d)))$$

```
if (Weather.w.Drivable.0,Day.v.Weather.w) then
(Delivered,Truck.t,Day.v) = 0
```

10. Higher prices mean lower sales

$$\forall d(\forall p \forall p'(p > p') \rightarrow Sales((p',d) > Sales(p,d))$$

```
if (Price.p as x > Price.q as y) then Sales,Day.v,Price.x > Sales,Day.v,Price.y
```

11. All US stores sell more veggie cakes than French stores

$$\forall s \forall s'(Location(s,US) \wedge Location(s',France) \rightarrow \forall d(Sold(d,s,Veggiecakes) > Sold(d,s',Veggiecakes)))$$

```
Count(Sales), Sales,Product.Veggiecakes,Location.US, Day.v as
x > Count(Sales) Sales,Product.Veggiecakes,Location.France,Day.x
```

12. Polluted ingredients in products imply polluted products

$$\forall i \forall p(Polluted(i) \wedge IsIn(i,p)) \rightarrow Polluted(p)$$

```
if (Polluted,Ingredient.i)=1 and (QtyIngredient,Product.p,Ingredient.i)
> 0 then (Polluted,Product.p)=1
```

13. Filler is in everything

$$\forall i(IsFiller(i) \rightarrow \forall p IsIn(i,p))$$

```
if (Filler,Ingredient.*.i=1) then (QtyIngredient,Ingredient.i,Product.*p
> 0)
```

14. There are products with no fish in them

$$\exists p \forall i IsFish(i) \wedge \neg IsIn(i, p)$$

```
if (IngredientType, Ingredient.*.i as x) = Fish then
(Count (QtyIngredient,Ingredient.x, Product.*.p = 0) > 0)
```

15. Polluted fish does not pollute everything (this is provable from the above)

$$\forall i(Polluted(i) \rightarrow IsFish(i))) \rightarrow \neg(\forall p Polluted(p)$$

```
Count (Product.*.p.Polluted.0,Ingredient.*.i.IngredientType.Fish.Polluted.0)
> 0,
```

16. Polluted filler does pollute everything (also provable from the above)

$$\forall i(Polluted(i) \wedge IsFiller(i)) \rightarrow \forall p Polluted(p)$$

```
if (Polluted,Ingredient.*.i.IngredientType.Filler = 1) then
(Polluted,Product.*.p = 1)
```