INTRODUCTION TO CAT4

Part 2. CAT2.

Andrew Holster[1]

Draft Jan 2021

Abstract

CAT4 is proposed as a general method for representing information, enabling a powerful programming method for large-scale information systems. It enables generalised machine learning, software automation and novel AI capabilities. It is based on a special type of *relation* called CAT4, which is interpreted to provide a semantic representation. This is Part 2 of a five-part introduction. The focus here is on defining key mathematical properties of CAT2, identifying the topology and defining essential functions over a coordinate system. The analysis is from first principles. This develops on from the axioms introduced in Part 1. The interpretation of *fact networks* is introduced in Part 3, and the full application to semantic theory comes in Part 4, where we introduce general functions, including the language interpretation or *linguistic functions*. In Part 5, we turn to software design considerations, to show how files, indexes, functions and screens can be defined to implement a CAT4 system efficiently.

[1] © Andrew Holster. 2020-2021. ATASA Research, Aotearoa New Zealand.

Contents

INTRODUCTION TO CAT4 0
Part 2. CAT2. 0
Introduction. 2
The CA2 relation. 2
The CAT2 relation. 4
Islands in CAT2. 5
Pushing atoms up. 7
The FCAT2 relation. 8
Absolute coordinates for FCAT2. 10
Absolute Coordinate Position Vectors. 12
Coordinate chains. 14
Transformations of coordinates. 15
The A-lattice. 16
Interior and Exterior geometry for positions. 17
Interior and Exterior geometry for points. 19
The Diamond Property. 20
Extension to classes of points. 23
General *Diamond* property. 24
Geometric Concepts. 26
CAT2 Topology. 28
Union and Intersection Classes. 29
The INTEXT Class. 31
Metric Space and Distance. 32
Composite Chains. 34
Shortest Chains. 35
GLB and Intersection Matrix. 39
Tree Symbols LTS and RTS. 42
Combinatoric problems. 49
Scalability and Interior Index. 50
Chains of CPs and Addresses. 55
References. 56

Introduction.

This continues from Part 1. Here we define restrictions on C2, and analyse properties which allow us to define general functions over a type of coordinate framework. The treatment here is by no means complete, but it is sufficient to go on with the subsequent development.

In Part 1 we defined C2. However our main interest will be in a more restricted class, having atomic sequences with unique points, i.e. no repetition of points in the atomic row of any interior. Note this means: *A≤D,* as there are only *D* possible unique points in a sequence of length *A.* E.g. for *D=3,* there are 15 possible sequences: *(1), (2), (3), (1,2), (1,3), (2,1), (2,3), (3,1), (3,2), (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1).* There are *D!/(D-A)!* unique sequences (permutations) of length *A* from *D* points. These correspond to interiors that are *lattices*, and we now define the *CA2 relation* as the sub-class of *C2 relations in which the interior of any point is a lattice.* Since every interior is now a lattice, the *point relation* corresponds uniquely to the *position lattice.* Hence, we can use the interior position coordinates for interior point coordinates. We subsequently define the stronger *CAT2 relation.*

The CA2 relation.

Axiom for CA2. A C2 relation is CA2 just in case all interiors are lattices.

Equivalent axiom.

Axiom for CA2. A C2 relation is a CA2 just in case all atomic sequences have unique points.

CA2 still allows atomic points in various orders within the same relation, e.g. we might add points: *(4,1,2)* and *(5,2,1).* The second is the *reflection* of the first. Or we may add: *(6,2,3)* and *(7,4,6),* defining the atomic sequence *1,2,3.* And also: *(8,1,3)* and *(9,5,8),* defining the atomic sequence: *2,1,3.* These may all be in the same CA2 relation.

0	0	0
1	0	0
2	0	0
3	0	0
4	1	2
5	2	1
6	2	3
7	4	6
8	1	3
9	5	8

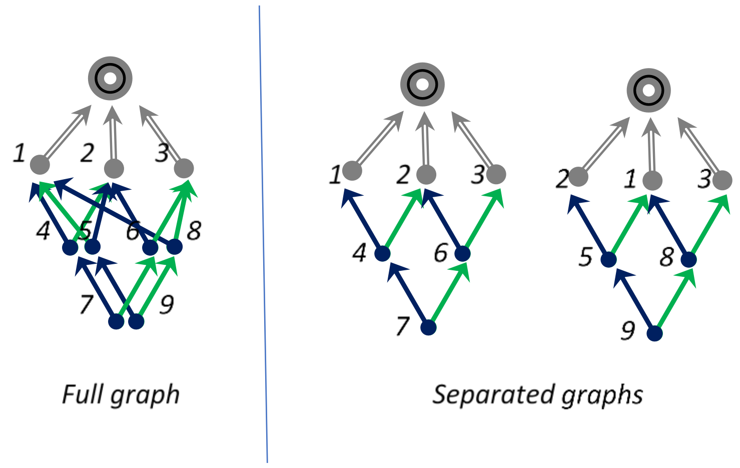


Figure 2.1. A CA2 graph (but not CAT2, below). This is the union of the two separate graphs on the right (which are separately CAT2).

However we cannot add a further point: *(10,4,5)* in CA2 (although it is permissible in C2) as this makes a circle (pipe) in the atomic sequence: *(1,2,1)* for the interior of the point *10.*

Similarly we can add: *(11,3,1)* and *(12,6,11)* but not: *(13,7,12)* at the fourth level as this has the atomic sequence: *(1,2,3,1).* Indeed, we cannot add any point at the fourth level in a CA2 relation with only 3 atomic points. In general:

- Proposition. A CA2 relation with *D* atomic points cannot have positions below level *D.*

This gives us a stronger structure than C2. However the combined graph above still looks messy, and is difficult to visualise clearly, unlike the simple position lattice. Furthermore, it is difficult to make *local recursive axioms* to ensure this structure, because we have to check multiple ordering possibilities. We now restrict attention further to graphs having atomic sequences with consistent order with each other. I.e. if sequences *A* and *B* are in the relation, then any pair of points *i,j* in both *A* and *B* occur in the same order at the same distance. E.g. if *123* is in the class, *{1,2,3, 12, 23, 123}* are all the possible seqs. This is the CAT2 relation.

The CAT2 relation.

Axiom for CAT2. A CAT2 relation is a CA2 relation where the atomic order is consistent for all interior lattices.

- Definition. The atomic order is consistent just in case the following condition holds.
- If two points *i,j* have the same CP, and have atomic sequences: $b_1, \ldots b_B$ and $c_1 \ldots c_C$, respectively (from the class of all atomic points a_i under the CP), then if $b_n = c_m$ and $b_p = c_q$, then: $p\text{-}n = q\text{-}m$.

This means that we can uniquely order the *atomic points,* a_i, for the CP. We can label points using an index system, e.g.

	z			Level 0 (CP)
A1	A2	A3	A4	Order categories
$a_{1.1}$	$a_{2.1}$	$a_{3.1}$	$a_{4.1}$	Level 1 (Atoms)
	$a_{2.2}$	$a_{3.2}$	$a_{4.2}$	
		$a_{3.3}$		
		$a_{3.4}$		

Figure 2.2. This represents a set of 9 atoms for the collection point CP=*z.* Atoms are now labelled as: $a_{l,m}$ where *l* is the (horizontal) *order* for the position, and *m* is a (vertical) *index* for multiple atoms in the position. Note the *Order Categories* are not part of the graph.

There can be any number of points in any column. The *order category* is the significant property. (The order of the *index* labels has no significance). Relative order is determined by points in the exterior. If two atoms, *j* and *k,* are joined below by *i*, i.e. *(i,j,k),* then *i* and *j* must be sequential in all interiors. I.e. if: $j = a_{l,m}$ then: $k = a_{l+1,n}$. The indices *m* and *n* are arbitrary, the important thing is that the order is sequential, so *l* and *l+1.*

However the order is determined by the exterior, which is contingent. We can have the situation where some atoms have no exterior, and hence no order determined relative to any other atoms. More generally, there may be two or more disconnected groups ('islands'), with no relative order relation determined between them. In this case there are multiple disconnected groups in the exterior, and the logical relations become complex. Hence order is not *determined* unless there are sufficient points in the exterior.

Islands in CAT2.

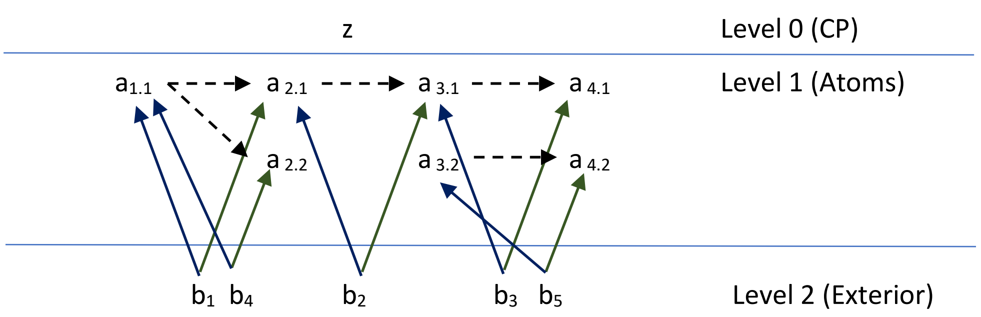


Figure 2.3. Illustration of two *islands.* *{$a_{3,2}$, $a_{4,2}$}* forms a small island disconnected from the other points.

Here the atoms in the top row are ordered by points *b_1, b_2, b_3,* in the next row and *$a_{2,2}$* is ordered relative to all points in that row by *b_4.* So *$a_{2,2}$* must be in the second order column, i.e. the same column as *$a_{2,1}$*. Also *$a_{3,2}$* is ordered relative to *$a_{4,2}$* by *b_5.* But there is nothing in the exterior to determine the relative order of the two groups – there are two *islands.*

But why should we label the second group in this position? It might be put anywhere, e.g.

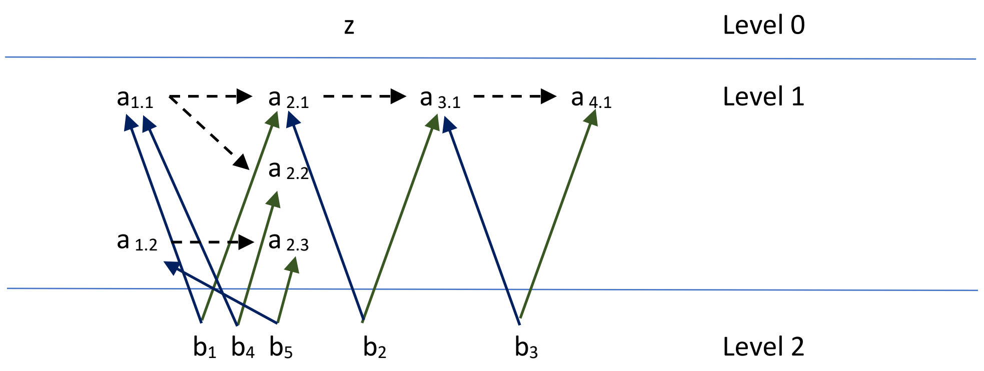


Figure 2.4. Atoms that were labelled *$a_{3.2}$* and *$a_{4.2}$* above are now labelled *$a_{1.2}$* and *$a_{2.}$*. I.e. we have labelled them in a different order relative to the other group. They are still ordered relative to each other by *b_5.*

Alternatively we might extend the atomic columns.

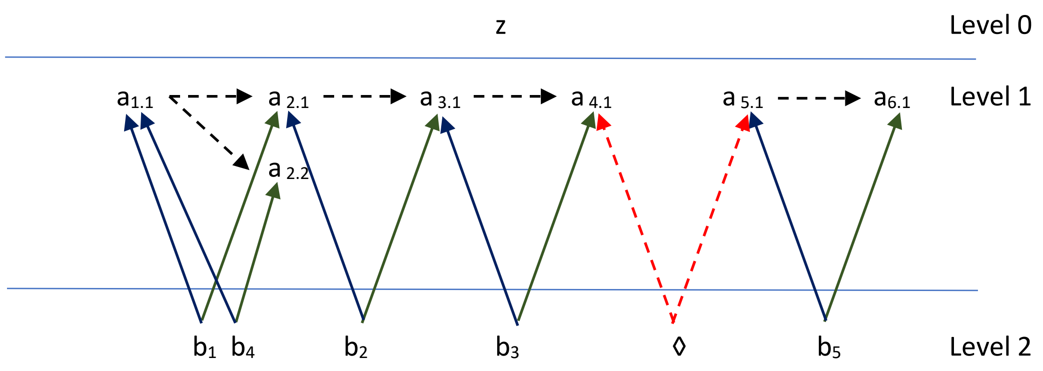


Figure 2.5. Atoms that were labelled $a_{3.2}$ and $a_{4.2}$ above are now labelled $a_{5,1}$ and $a_{6,1.}$ We have extended the order categories to the right. It is now possible to add a new point ◊ in this ordering.

Or alternatively again, we might place these two points to the left of the main group – and then we would need to relabel the *order columns* of the other points so the order labels start with 1.

If we want to assign atoms to order categories, what determines the choice? It must be consistent at least with the other points present lower down. But the order of the small island is not determined at all by anything in the current graph: it depends on what we want to use these atoms for in the future. If this small island is never going to have any exterior relations to the main island, it doesn't matter, and we could have a convention that we introduce all independent islands on the left. Note in this case that there can never be joins between the two islands, they have no relations except through the CP, and we could separate two such islands under two distinct CP's. The two CPs will be in the same positions in the larger graph. But this would remove the future *possibility* of relating them in an extension of the graph.

But if we do want to join them to the other island in the future, it depends what points we want to be able to join them to. E.g. the order above is necessary *if we want to add a point below joining $a_{4,1}$ to $a_{5,1}$.* The position indicated as ◊ is consistent with this ordering, but not consistent with the previous orderings.

We may also ask: what *represents* the ordering in the graph? The *order labels* are not part of the graph: they are just a way for us to talk about it. They provide us with structured variables over the points. The only determination of order in the *graph itself* is through the exterior points. Until we add a specific exterior point joining *islands* to each other, we are free to join them in any relative order. And we might also remove such points in the future, and lose the relation between islands, so the order information represented by the exterior is not very robust.

This is a difficulty in assigning these order labels as variables over points – we can only assign the variables over single *islands* at a time. The variables for different islands do not relate to each other. But we ideally want to use these labels as variables to refer over all the points.

Having multiple atoms in the order positions, and allowing order to be indeterminate until they are related by exterior points, makes things formally more complex, and the next step is to simplify the system by a simple technique that does not reduce the representational power. This is called *pushing the exterior (or atoms) of the CP up a level.*

Pushing atoms up.

We introduce an extra level between the CP and the present atoms, with a single unique sequence that becomes the new atomic level.

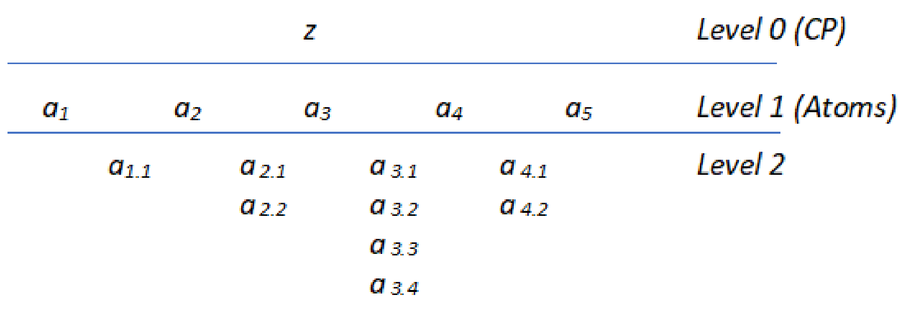


Figure 2.6. We introduce a sequence of atomic points, a_1, a_2, a_3, a_4, a_5, which now represent the *order columns* from the previous diagram. These are now points in the graph, with double-joins to z. The original atoms have been 'pushed up' a level (down in the diagram).

We now have a sequence of five atoms instead of four. This now explicitly defines the order of the original atoms, determined through their interiors. Previously this order was only implicit, through the extensions. Atoms without extensions, or islands, had no order previously, but now all points are defined with an explicit order from the atoms.

However we still have the situation that the *ordering* of the new atoms is now represented only by the exterior, at Level 2. If there are *A* atoms, there must be a sequence of *A-1* points in the second row (the original atoms) joining them for the atomic level to be ordered.

Now we cannot join any points in the second row without determining an order relative to the new atoms. And it must be consistent with any extension below in the third row. Nonetheless when there are islands, there is still an open choice of relative order of the two islands. However the graph must now embody a specific choice of order relations – we cannot leave it free as before. So this enforces a stronger structure – but still lets us specify any graph structure below the atomic row that we could specify before.

We now visualise in terms of the process of building a graph. We first specify the number of atoms we need in the first row. This determines how many Levels the graph can contain. (We can always extend it of course). The atoms are introduced only to represent order and the size of the possible lattices. So their identity is irrelevant in a sense: they do not represent any information other than this order. But we must define an order through points at Level 2. Instead of inserting just these points, we insert a full interior lattice that orders the atoms, determining the lattices. This acts like a coordinate system. Indeed, we can insert such points with the sole purpose of providing a *coordinate lattice.* This has no other joins to other points. This is exactly like the simple flat graphs illustrating coordinate systems we saw earlier.

We define a local CAT2 space by inserting a lattice extending across the entire set of atoms. Remember this is generally a sub-space within a larger CAT2 relation, although it can be a complete CAT2 relation by itself, if we make the CP point the zero point.

This defines the 'space' of the graph so to speak, making it unambiguous. We can then insert any other points in the *positions* defined by the coordinate lattice. We can embed the original graphs we saw at the start of this section on top of this space. But now the original graphs are pushed up a Level. If we define every graph with a *coordinate lattice* we have a simple form for defining graphs. We can now create a complete coordinate system covering positions for the whole exterior of the CP. These may now be labelled from the top down, instead of bottom up (later). We call these flat CAT2 lattices.

The FCAT2 relation.

We define *a flat CAT2 relation* or *FCAT2*, as follows.

Axiom for FCAT2. An *FCAT2* relation is a CAT2 relation where all CP's it contains are flat.

- Definition. A *CP* is called *flat* just in case its atoms are uniquely ordered in every interior.

This is the critical restriction on the general CAT2 relation for the database application. We primarily want to use *flat CAT2s.* (Whether we allow some other structures is still an open question). But we cannot restrict a dynamic system to flat CAT2 relations, since discrete transformations defined by adding or removing single points creates intermediate states that are not flat (although the initial and final states may be Flat CAT2).[2]

- Proposition. To extend a flat CAT2 relation by discrete operations, we generally must have intermediate relations which are not flat.
- Proof. Suppose we add two atoms (i.e. points with double-joins) below a point. Until we add a coordinate point that orders them, it is not a flat CAT2 (just CAT2).
- We cannot add the order point until the two parents are present.
- Therefore the sequence of relations required to add a CP with even two atoms must include a non-flat relation.

However although the flat *FCAT2 relation* is more restricted than the CAT2 relation, this does not restrict the representational power, because any CAT2 relation can be embedded into a flat CAT2 relation with the same representational power.

- Proposition. Every CAT2 relation has a homeomorphic embedding into a flat CAT2 relation.
- Proof. First show that this holds for the exterior of any CP within a CAT2 relation. Then prove inductively that it holds for whole CAT2 relation.
- Take a CP and its exterior in a CAT2 relation. Suppose it has the general structure as illustrated above, with multiple atoms in order categories determined by the exterior, and multiple islands, so relative order is not always determined.
- Transform by *pushing up.* Define the number of atoms as the longest sequence of ordered atoms plus 1. Order all independent islands from the left (by convention).
- Then show this contains the original structure.
- The homomorphism is defined by mapping each point in the extended relation to itself, and each join to the original join. Note the joins from the original atoms to the CP no longer correspond to single joins in the extended relation, but instead to chains to the CP.
- When this is done for every CP, we have a flat CAT2 relation.

[2] Note in a database context, this might be managed with *transactions,* which are multiple operations defined in sequence, executed in one action. Typically we perform a transaction, check if the result is consistent with our rules, and *commit* if it is, or *roll-back* if not. However we do not want to make this limitation on fundamental operations, and we allow discrete operations.

Note however that if we have two islands initially in a CAT2 relation C_2, and push up, by our convention they come out ordered on the left in a flat CAT2, call it F_2. If we subsequently joined them in the original CAT2, creating a new relation C_2' and push up, we get F_2'. However we cannot generally add the new point to F_2 to get F_2'. We may order islands in incompatible ways in C_2', and we would need to reposition them in F_2 to get F_2'. A second method would be to push up separate islands under separate CP's. But if we subsequently wish to join them, we would then need to move them back under the atoms of one CP.

This raises the general issue that for a dynamic system we need to think about the process of *moving lattices around.* I.e. not just adding, deleting or moving single atoms at once, but whole blocks. Again this relates to the process of transforming relations through a sequence of intermediate states. We cannot do this one point at a time and retain CAT2 at every stage when there are multiple points in an interior to move. Note we can always avoid moving points by duplicating them as new points in the new positions (copying), and then deleting the originals. But this will be very inefficient with large graphs. There may be millions of points to move, but generally only a handful of atoms. And it also means renumbering points because we cannot have duplicates in the primary key.

Absolute coordinates for FCAT2.

Exteriors of CPs in a FCAT2 can now be labelled with top-down coordinates. Note this *canonical coordinate system* takes Levels as *increasing* downwards, while the interior coordinates took Levels decreasing downwards. It conforms to the vector algebra of the interior coordinates, but they now provide a system of *absolute coordinates*, which removes the need to calculate interior vectors. This is possible because the FCAT lattice covers the whole space.

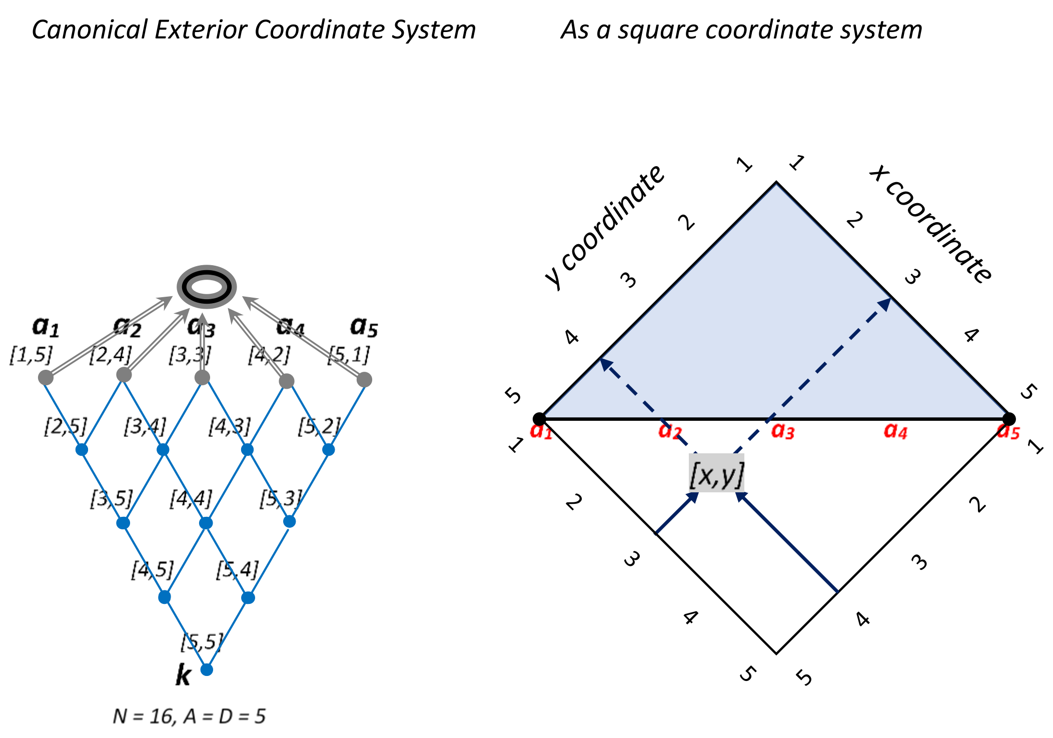


Figure 2.7. Left. Illustrating top-down exterior coordinates for *A = 5*. This is symmetric w.r.t. left-right reflection, and has no negative numbers. Right illustrates how this corresponds to discrete coordinate axes, *X,Y,* as for a grid. We use only the bottom half-triangle of the coordinate plane. The top half (blue) only represents the CP. We will assign this the coordinate *[0,0].*

We define the *coordinate function* for points *p* as:

- Definition. Coordinate function for points.
- *[p] = [x,y]* where: *[x,y]* is the 2-place coordinate for point *p.*
- *x, y are integers.*

If: *[p] = [q]* for points *p,q,* they are at the same coordinate position. But they need not have the same parents. (Points *p* and *q* with *identical parents* are said to be in the same *position in the graph,* which is stronger than having the same coordinate position.)

We can formally define canonical coordinates as follows.

- Definition. Canonical coordinates for points in an FCAT lattice above the CP.
- Atomic points ordered: *(a_1,…a_i, … a_A)* from left to right have coordinates: *[(i),(A-i)].*
- If a position *[x,y]* has a right child (i.e. down-right), the child coordinates are: *[x+1,y].*
- If a position *[x,y]* has a left child, the child coordinates are: *[x,y+1].*

We might extend the rule to assign coordinates *upwards.*

- If a position has coordinates *[x,y],* the *left parent position* has coordinates: *[x-1,y]* and the *right parent position* has coordinates: *[x,y-1].*

This gives identical coordinates up to the atomic row, but above that we now get multiple coordinates, which correspond only to the CP position. To allow a unique coordinate for the CP, we define:

- Definition. The CP position has coordinates *[0,0]*
- Definition. Any coordinate *[x,y]* where $x+y < A$ is equivalent to *[0,0]*

We define the *Levels* so that the atoms are at level 1, and increase going down the lattice:

- Definition. Levels for canonical coordinates.
- The Level of a position above the CP with coordinates: *[x,y]* is: $L[x,y] = x+y-A$.

This defines Levels in the reverse order to the interior coordinate system. I.e. going up one step means decreasing *L* by one. The vector algebra is otherwise equivalent to that for the interior coordinate system.

Absolute Coordinate Position Vectors.

- Definition. Coordinate vectors below a CP.
- If $[i] \equiv [x_i,y_i]$ and: $[j] \equiv [x_j,y_j]$ are position coordinates within a CP, not including *[0,0]*, the *vector from [i] to [j]* is: $\mathbf{v}_{i,j} = [x_j-x_i,y_j-y_i]$.

The main issue is that the class of *vectors connecting positions* is constrained by the size *A* of the CP. E.g. there are no vectors connecting positions larger than: $\mathbf{v} = [A-1,0]$ or smaller than: $\mathbf{v} = [-A+1,0]$. Or generally if: $\mathbf{v} = [x,y]$ then: $|x+y| < A$.

However we can define the vectors with components as ranging over any integer values. Then vectors generally lie outside of *positions,* and we must constrain the addition of positions to vectors: *position + vector → new position* to cases where the new position exists. However the class of vectors can be treated as the full class.

- Definition. Vector addition.
- $\mathbf{v}_j + \mathbf{v}_j = [x_i,y_i] + [x_j+y_j] = [x_i+x_j,y_i+y_j]$ if: $|x_i+x_j+y_i+y_j| \leq A$
- Definition. Scalar multiplication.

- *a**v** = a**[**x,y**]** = **[**ax,ay**]*** if: *|ax+ay| ≤ A* and *a* is an integer.

We can add a vector to a position to get to another position:

- Definition. Position-Vector addition.
- *[x,y] + **[x',y']** = [x+x',y+y']* where: *0 ≤ (x+x') ≤ A* and: *0 ≤ (y+y') ≤ A* and: *A < (x+x'+y+y') ≤ 2A,* otherwise: *[x,y] + **[x',y']*** is null (no position).
- Equivalently, *[x,y] + **[x',y']** = [x+x',y+y']* exists only when: *–L([x,y]) ≤ x'+y' ≤ A – L([x,y]).*

Remember the *Level* of a position is: *L([x,y]) = (x+y – A).* We define the Level of a *vector as:*

- Definition. Level of a Vector. *L**[x',y']** = x'+y'.*

So if we add a vector position to a vector: *[x,y]-**[x'y'],*** the levels add:

- *L([x,y]-**[x'y']) =** L([x-x',y-y']) = (x+y-x-y-A) = L([x,y])-L(**[x'y']**)*

Note that:

- Proposition. Vectors are independent of linear transformations of the coordinate system.

Note the *positions* stop at the atomic row. The CP will have its own exterior coordinates defined by its position in its own CP. We may define the coordinates of the CP as: *[0,0],* and its Level as 0.

- Definition. *[CP] = [0,0]*
- Definition. *Level[0,0] = L[0,0] = 0*

Then position + vector addition holds for all positions in the CP, but not the CP. For these *vectors* to form a vector space they must satisfy these axioms.

Axiom	Meaning
Associativity of addition	$\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$
Commutativity of addition	$\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
Identity element of addition	There exists an element $\mathbf{0} \in V$, called the *zero vector*, such that $\mathbf{v} + \mathbf{0} = \mathbf{v}$ for all $\mathbf{v} \in V$.
Inverse elements of addition	For every $\mathbf{v} \in V$, there exists an element $-\mathbf{v} \in V$, called the *additive inverse* of $\mathbf{v}$, such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$.
Compatibility of scalar multiplication with field multiplication	$a(b\mathbf{v}) = (ab)\mathbf{v}$
Identity element of scalar multiplication	$1\mathbf{v} = \mathbf{v}$, where 1 denotes the multiplicative identity in F.
Distributivity of scalar multiplication with respect to vector addition	$a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$
Distributivity of scalar multiplication with respect to field addition	$(a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$

[Wikipedia. Vector space].

The scalars, *a,b,* are the *integers.* These axioms are satisfied as long as we allow vectors of any size. Thus we have the same *vector space* for all CP's, but distinct *position spaces* for different sized CP's. We can take the *position spaces* of CP's to be contingent states (or initial conditions), imposed on the background *vector space,* forming 'containers' in which to place the *points.* We subsequently impose a distribution of points on the position space, which is analogous to imposing a contingent distribution of bodies on a Euclidean space. However there is a major difference, in that bodies in Euclidean space have all their spatial relations determined by the Euclidean position relations, whereas points in C2 space do not have all their graphs relations determined by the C2 position relations. Another difference is that we now have a *manifold of CP points* with their separate position spaces. The manifold of CPs is a tree, and relations between points in different CPs are determined by these trees. The trees can be described with *coordinate chains.*

Coordinate chains.

Note that since the exterior coordinates are relative to a CP we may index coordinate functions in *coordinate chains*. The following takes *z, z', …,* to be the numbers for CP points, and uses: *[z], [z'], etc,* to construct bracketed symbols for those points.

- Definition. Coordinate chains.
- *[0][x,y][z][x',y'][z'] … [x'',y''][z''][x''',y''']* represents a complete coordinate chain from the zero point *[0]* to the final position in the chain: *[z''][x''',y'''].*
- This position exists just in case the sequence: *[0][z']…[z'']* is the complete sequence of CPs in the interior of *[z''],* and the position: *[z''][x'',y'']* exists.

We may say that coordinate chains are *valid* only if they have appropriate coordinates for the CPs.

- Definition. Valid coordinate chains.
- A coordinate chain is *valid* iff all its terms are valid, as defined below.
- A coordinate chain containing: *…[z][x,y]…* is *valid* only if point *z* has at least: *x+y-A* atoms.
- A coordinate chain containing: *…[z][x',y'][z']…* is valid only if *[z]* is the CP of *z'.*
- A coordinate chain containing: *…[z][x',y'][z']…* is valid only if [z'*]* is at position *[x'y']* in *[z].*
- A coordinate chain is *complete* only if it begins at *[0]* and contains the complete sequence of CPs up to the final coordinate position.

Coordinate chains are a more complete basis for *coordinate vectors* going across the entire graph.

- Proposition. Each point p has a unique complete coordinate chain: *[0]…[x,y].*

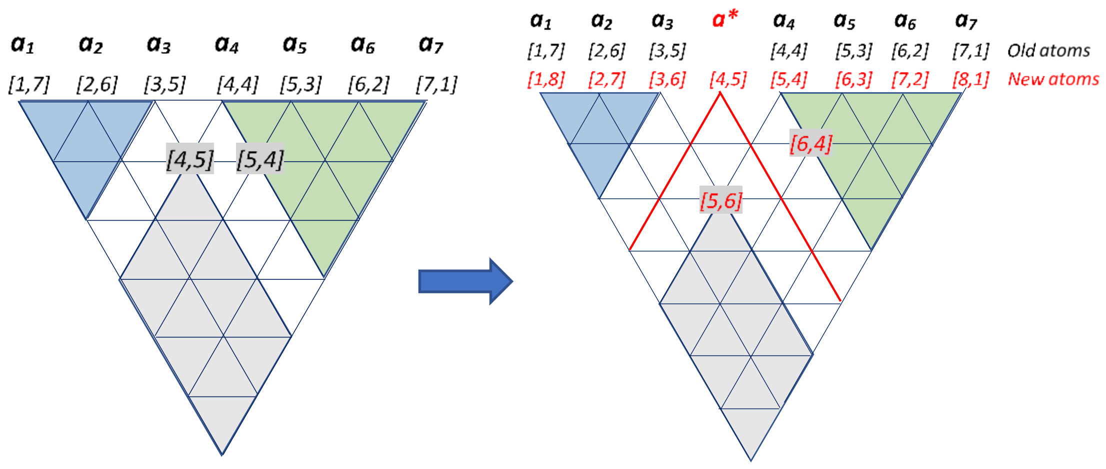
Transformations of coordinates.

The correspondence to the *XY* grid on the right illustrated above shows why the canonical system gives the most symmetric coordinate system. There are several equivalent variations of this coordinate system, equivalent to *translation* and *reversal* transformations, e.g. we might define any linear combination:

- Definition. Linear transformations of coordinate systems.
- Map: $[x,y] \rightarrow [x',y']' = [\pm x+X_0, \pm y+Y_0]$ where X_0,Y_0 are constants, to form an isomorphic coordinate system.

Coordinates must change if we extend the atomic row. We may insert a new atom *a** between two existing atoms at: $[a_n]\equiv[n,A-n+1]$ and $[a_{n+1}]\equiv[n+1,A-n]$. (We allow $n \varepsilon [0,A]$ here to insert to the left of a_1 or to the right of a_{A-1}.) The transformation may be defined piecewise as:

- Definition. Transforming coordinates by adding an atom.
- If *[x,y]* is an original position, there are three cases:
    - Case A. $x \le n$ and $y \ge A-n$. Then: $[x,y] \rightarrow [x,y+1]$
    - Case B. $x \ge n$ and $y \le A-n$. Then: $[x,y] \rightarrow [x+1,y]$
    - Case C. Otherwise. Then: $[x,y] \rightarrow [x+1,y+1]$



a₁ a₂ a₃ a₄ a₅ a₆ a₇
[1,7] [2,6] [3,5] [4,4] [5,3] [6,2] [7,1]
[4,5]
[5,4]
a₁ a₂ a₃ a* a₄ a₅ a₆ a₇
[1,7] [2,6] [3,5] [4,4] [5,3] [6,2] [7,1] Old atoms
[1,8] [2,7] [3,6] [4,5] [5,4] [6,3] [7,2] [8,1] New atoms
[6,4]
[5,6]

Figure 2.8. Left is a graph with 7 atoms. On the right, an additional atom is inserted between points *3* and *4.* This makes 8 atoms and introduces 8 new points (red lines). All existing points must have coordinates relabelled since the atomic number has changed, and 6 points have new parents.

There are other methods to extend the lattice, e.g. extending on the right and moving only selected points across, etc. This illustrates the general problem.

The A-lattice.

To ensure the FCAT structure, we can explicitly define a *complete coordinate lattice* for every CP. We will refer to this as the *A-Lattice* (for atomic lattice). For any point *z* with a set of *A>1* atoms (double-joined points below), we can define a lattice spanning them.

- In the database context we can do this physically, constructing the A-lattice as the first step before constructing exterior graphs (the “empirical graphs”). We make no other C2 joins to these, so they never get mixed up with the rest of the graph.[3]
- In the mathematical context, we can consider this as a virtual graph that we can refer to, distinct from the empirical graph.

[3] Note in the database application, we want to distinguish the A-lattice from the ‘empirical lattice’, so we do not get it mixed up, e.g. in counting classes, etc. There is nothing in C2 joins to make this distinction, but it is done with the third or fourth joins.

The *A-lattice* looks just like the pattern of the canonical coordinate graph above, with the assumption that it spans the entire set of atoms. Thus every CP *z* has an *A-lattice* defined as follows.

- Definition. A-lattice for a CP *z*.
- Assume the class of atoms below *z* is: $(a_1, \ldots a_n, \ldots a_A)$. *A* is the number of atoms.
- Step 0. Label the atoms with indices: $a_{n,(A-n+1)}$. This forms the sequence: $(a_{1,A}, a_{2,A-1}, \ldots a_{A,1})$. These are Level 1.
- Recursion step. Add a point-relation: $(i,j,k) = (i, a_{x-1,y}, a_{x,y-1})$ for every pair: $(a_{x-1,y}, a_{x,y-1})$ in the previous Level. Label these with indices: $a_{x,y}$. These are Level *x+y-A.*
- Repeat until Level *A.* The A-lattice is complete.

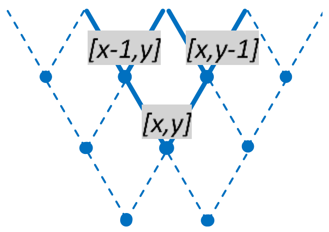


Figure 2.9. Coordinate labels at an arbitrary point in the coordinate lattice.

- Exercise. Show this method of labelling coordinates is equivalent to the canonical system.
- The number of points in the A-lattice are the triangular numbers: $A(A-1)/2 \rightarrow A^2/2$.

Interior and Exterior geometry for positions.

Referring back to the earlier diagram of interior and exterior sets, we now show how the position coordinates gives an efficient way to formalise intersections and unions of interior and exterior sets. We introduce the following functions.

- Let *p* and *q* be any two points with the same CP in a C2 relation.
- Assume points *p* and *q* have position coordinates: $[p] \equiv [x_p, y_p]$, $[q] \equiv [x_q, y_q]$.
- We define these operators.
- Minimum: $min(x_p,x_q)$ *and:* $min(y_p,y_q)$
- Maximum: $max(x_p,x_q)$ *and:* $max(y_p,y_q)$
- Interior Meet position: $[p]$^$[q]$
    - $[x_p,y_p]$^$[x_q,y_q] = [min(x_p,x_q), min(y_p,y_q)]$
    - Note if: $min(x_p,x_q) + min(y_p,y_q) < A$ then $[min(x_p,x_q), min(y_p,y_q)] \equiv [0,0]$, the CP.

- Exterior Join position: *[p]v[q]*
    - $[x_p,y_p]v[x_q,y_q] = [max(x_p,x_q),max(y_p,y_q)]$
- Left position: *[p]<[q]*
    - $[x_p,y_p]<[x_q,y_q] = [min(x_p,x_q),max(y_p,y_q)]$
- Right position: *[p]<[q]*
    - $[x_p,y_p]>[x_q,y_q] = [max(x_p,x_q),min(y_p,y_q)]$

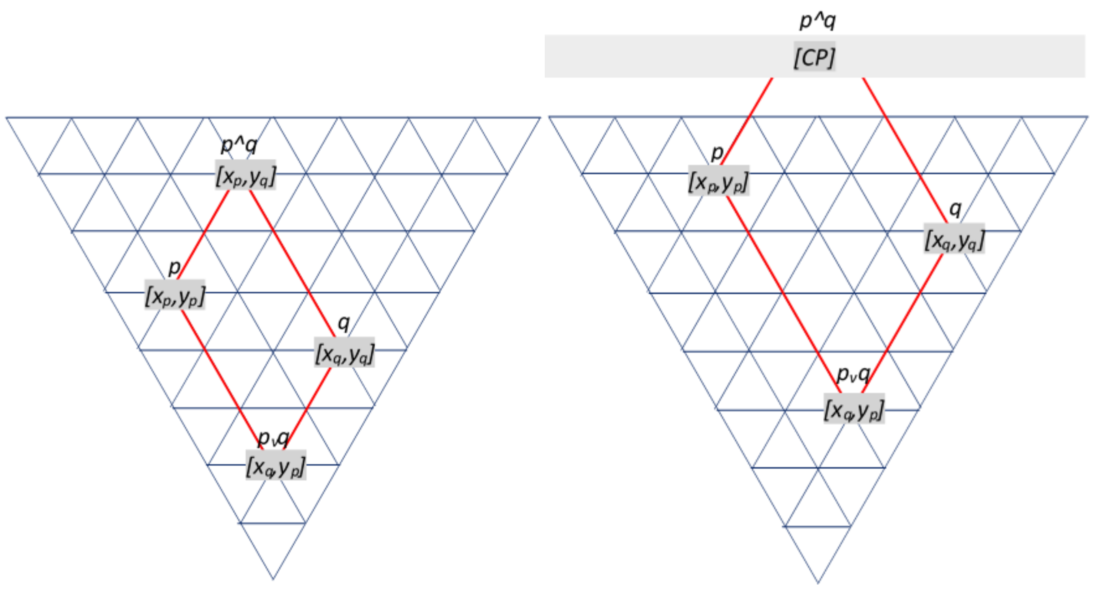


Figure 2.10. Illustrating *[p]^[q]* and *[p]^{v}[q]* when *p<q.* Left shows two points with *[p]^[q]* in the interior. Right shows two points where *[p]^[q] = [0,0],* i.e. the CP position. The latter occurs whenever two points are sufficiently separated horizontally.

If *p* is in the interior of *q,* the meet and join positions are *[p]* and *[q]* respectively. Every pair of positions in the same CP has a meet and join position in the CP (including the CP).

We define *interiors of positions* by:

- Definition of Interior Positions.
- $INT[x_p,y_p]$ is the class of all positions: *[x,y]* such that: $x \le x_p$ and: $y \le y_p$.
- Note valid position are where either: $x+y > A$, or $x=0$ *and* $y=0$ (for the CP).
- *[CP] = [0,0]* is in the interior of all positions of the CP exterior.
- Note we can take positions: *[x,y]* where $x+y \le A$ to be identical to: *[0,0],* and then we can allow *x,y* to range over *0* to *A,* with the constraint on valid positions above.

Exterior positions are defined as the inverse of Interior positions, i.e.

- Definition of Exterior Positions.
- $[x_q, y_q] \varepsilon EXT[x_p, y_p]$ just in case $[x_p, y_p] \varepsilon INT[x_q, y_q]$
- Equivalent Definition. $EXT[x_p, y_p]$ is the class of all positions $[x,y]$ such that: $x \geq x_p$ and: $y \geq y_p$, and either: $A < x+y \leq 2A$, or $x=0$ *and* $y=0$ (for the CP).
- Note $[CP] = [0,0]$ is in the exterior of $[0,0]$.

Interior and Exterior geometry for points.

The previous section defines functions on *positions*, not on points at positions. For the corresponding *points,* we define the *Meet p^q* of points *p,q* as the (GLB) point *r* in the interior intersection of *p* and *q* at the interior join position if such a point exists, otherwise null.

- Definition. Meet *p^q.* This is a unique point, or null.
  p^q = r if *[p]^[q] = [r]* and *r is in INT(p)* and *r* is in *INT(q).*
- Otherwise *p^q* is null.

We similarly define the *Join:* $p^{\vee}q$ of *p,q* at the *LUB* position.

- Definition. Join $p^{\vee}q$. This is a class of points, or null.
  $r \, \varepsilon \, p^{\vee}q$ if *[p]v[q] = [r]* and *r is in EXT(p)* and *r* is in *EXT(q).*
- Otherwise, if there are no such points, $p^{\vee}q$ is null.

These operations can be extended beyond a single CP, using coordinate chains. However we first work within the CP, i.e. at coordinate positions: *[z][x,y]* where *z* is the CP.

Key propositions relate interiors of points to interiors of positions. Interior positions of a point *q* correspond 1-1 with interior points.

- Propositions. If point *p* is in the INT of point *q,* then position *[p]* is in the INT of position *[q].* Ditto with EXT. Formally:
- $p \, \varepsilon \, INT(q)$ just in case $[p] \, \varepsilon \, INT([q])$.

Interior *points* are unique w.r.t interior *positions*, i.e. every interior position of a point corresponds to a unique interior point. But *exterior points* are not unique, i.e. to every exterior position of a point, there may be multiple exterior points, or there may be no exterior points.

- Proposition. The highest possible level of a point (GLB) in the intersection of the interiors of two points: $INT(P) \cap INT(q)$ is at: *[p]^[q].*

- Proof. The Level is defined as: *x+y-A.* Interior positions of *p* and *q* are at *[x,y]* where: $x \leq x_p$, $x \leq x_q$, $y \leq y_p$, $y \leq y_q$. Let: $x_s = min(x_p,x_q)$ and: $y_s = min(y_p,y_q)$. Then there are no interior positions of both *p* and *q* higher than: $[x_s,y_s]$, since this requires: $L = x_s+y_s < x_t+y_t$, but then either $x_s < x_t$ or: $y_s < y_t$, but this contradicts the previous relation.

Note that the position: *[p]^[q]* is the highest possible interior intersection position of *p* and *q,* but there is not necessarily a *shared point* at that position. (There is a shared a point only if they *meet.* If they meet, they can be *joined* in the exterior, forming a lattice.)

Note viewing CAT2 as a *lattice,* we define the concepts of Greatest Lower Bound and Least Upper Bound of a pair (or class) of positions or points. The GLB of *p, q* is the *highest point in the intersection of interiors*. It is equal to *p^q* if a point exists there, in which case we may say *p* and *q* are in the same lattice. Otherwise it is class of highest-level points of: MAX(*INT(p)∩INT(q)*).

If they do not *meet* at: *[p]^[q],* we say the *meet: p^q* does not exist. We also say the points are *split.*

They must then still *intersect* at some point/s at a (lower) Level, say: (*[p]^[q])*-***[x',y']*** where: $0 < x'+y' \leq L([p]^[q])$.

The Diamond Property.

A key result is summarised here as the *Diamond Property*. This lets us find graph solutions using coordinate algebra.

- The *Diamond property* means: If two points *p* and *q* in a C2 relation have any shared points in their exteriors, then:
  - (A) they must have a unique shared point in their interiors, *p^q = r* at the unique (GLB) position: *[p]^[q],* and
  - (B) the intersection of their exteriors is the exterior of the intersection: *EXT(p˅q),* at the unique (LUB) position: *[p]˅[q].*
- Formally. If: *EXT(p) ∩ EXT(q)* is not empty, then (A) there is a single point: $r \varepsilon INT(p) \cap INT(q)$ which is at (GLB) position: *[p]^[q],* and (B) *EXT(p) ∩ EXT(q) = EXT(p˅q).*

This is a general property of C2, but we illustrate it for CAT2. The reason can be seen from the following diagram.

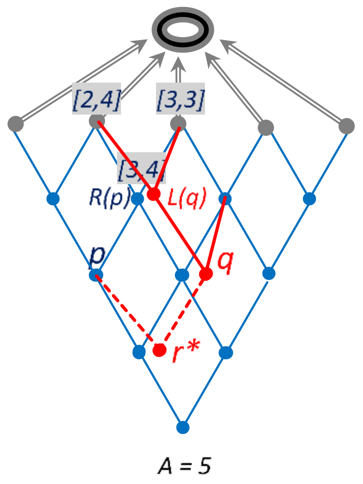


Figure 2.11. The points *p* and *q* are both valid, but they cannot be joined below by *r** (which is not a valid point), because the *LR* and *RL* grand-parents are split. Because the parents are split and do not intersect, the structures below *p* and *q* foliate in separate 2-D layers. They cannot be joined again in CAT4.

The points *p* and *q* can never be joined by a shared exterior point (below), as shown by *r*,* because the parents of *p* and *q* are split. *LR(r*) ≠ RL(r*),* which contradicts C2. The interiors of points *p* and *q* do not intersect at position: *[p]^[q] = [3,4]*, but rather at two positions: *[2,4]* and *[3,3].* The intersection of their interiors: *INT(p) ∩ INT(q)* in this example starts (GLB) at level *1,* with two points, not at level *2* with a single point.

By extension, *p* and *q* (and any children of *p* and *q*) can never join below no matter how far the graph is extended. Hence there can be no shared points in the exteriors, and: *EXT(p)∩EXT(q)* is empty.

Conversely, the Lattice property tells us there if is a common point in the exteriors of *p* and *q*, call it *r*,* then since the interior of *r** is a lattice, any pair in it including *p* and *q* must join above at a single point, *p^q.* This is the defining property of a lattice. The following is another demonstration using the coordinate geometry.

If points are split they can never be joined. To give a general proof we can give an inductive proof, or use the coordinate algebra, as follows.

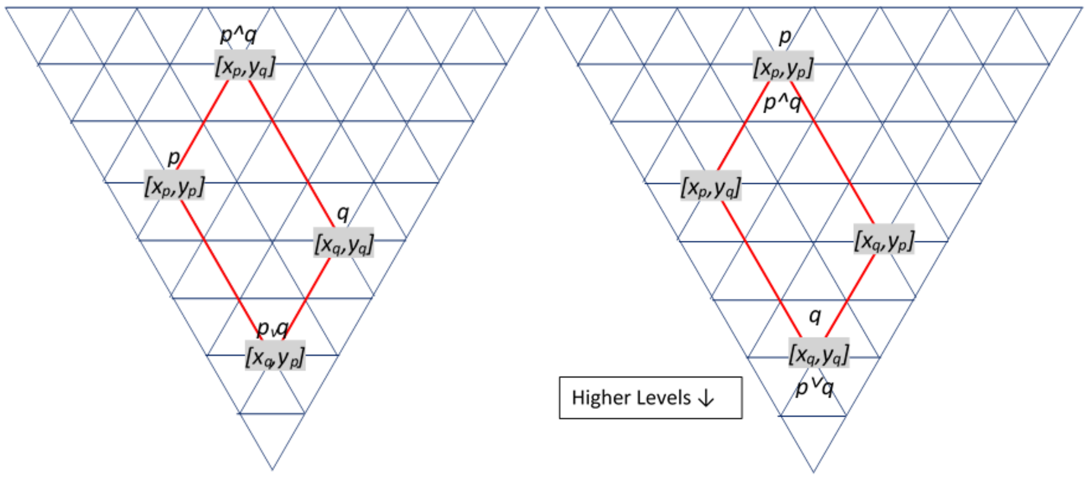


Figure 2.12. The two cases. Left diagram. *p* is left of *q.* They are not in each other's interior or exterior. There may be a common point in their exteriors, but (*Diamond property*) only if there is a common point: *p^q* in their interiors. Right diagram. *p* is in the interior of *p* and *q.* (*p* is at a lower Level than *q.*) *q* is in the exterior of *p* and *q.* Cf. the previous diagram shows the special case when *p^q* is the CP point. This special case conforms to the same algebra.

- Proof of *Diamond property*. There are four possible relationships between *[p]* and *[q], viz.*
  - 1(a) $x_p \leq x_q$ and: $y_p \leq y_q$
  - 1(b) $x_q \leq x_p$ and: $y_q \leq y_p$
  - 2(a) $x_p \leq x_q$ and: $y_q \leq y_p$
  - 2(b) $x_q \leq x_p$ and: $y_q \leq y_p$
- *1(a)* is equivalent to *1(b)* by exchanging the names of *p* and *q,* and similarly *2(a)* is equivalent to *2(b)*. Hence, we may analyse just the two cases, *1(a)* or *2(a).* These are illustrated above.
- First Case. $x_p \leq x_q$ and: $y_p \leq y_q$. Then by definition of interior positions, *[p]* is in the interior of *[q]* and *[p]= [p]^[q]* and: *p = p^q.* Hence *p* is in the interior of *q.* Also, for any point *p, p* is the unique lowest point in *INT(p)*. Hence *p* ε *(INT(P)∩INT(q)),* and *p = p^q,* and *p* is the unique lowest point in *(INT(P)∩INT(q)).* Hence the property holds for this case.
- Second case. $x_p \leq x_q$ and: $y_q \leq y_p$. Assume there is some *r* ε *EXT(p)∩EXT(q)* with *coordinates: [r] =* $[x_r, y_r]$. By our previous propositions, $x_p \leq x_r$ and: $y_q \leq y_r$. Then *r* has an interior point, *r′,* at position: $[x_p, y_q]$. This is the position: *[p∨q].* This is the lowest (level) position in the shared exterior of *p* and *q.* Then *r′* also has an interior point, *s,* at: $[x_q, y_p]$. This is the same position as: *[p]^[q].* This is also an interior point of both *p* and *q.* Interior points are unique (for each position). Hence *s* is a unique point in the interior of both *p* and *q.* The

position: *[p]^[q]* is also the lowest possible position common the interiors of *p* and *q* (proposition above). Hence the property holds for the second case.

- Hence the property holds for all cases.

The *Diamond property* is important in applications, used to calculate interior and exterior intersections. *Diamond property* is closely related to these:

- *INT(p) ∩ INT(q) = INT(p^q)*
- *EXT(p) ∩ EXT(q) = EXT(p˅q)*
- *EXT(p^q) ≥ EXT(p˅q)*

Extension to classes of points.

We can generalise the previous functions to act on classes of points. We have defined the functions: *[p]^[q], [p]˅[q], p^q* and *p˅q* as binary functions. We can then consider: *([p]^[q])^[r], ([p]˅[q])˅[r], (p^q)^r* and *(p˅q)˅r.* These are both *commutative and associative.* Note also that for the A-lattice alone, the positions and points have a 1-1 correspondence. Hence if we can show algebraic properties for *points* this shows the analogous property for *positions,* since positions correspond to a special case of points. Hence proofs of the following for *points* can be considered as more fundamental than proofs for positions.

- Proposition. The operations ^ and ˅ are commutative.
- For positions: *([p]^[q]) = ([q]^[p])* and: *([p]˅[q]) = ([q]˅[p])*
- For points: *(p^q) = (q^p)* and: *(p˅q) = (q˅p)*
- Proof. Set intersection is commutative so: *INT(p) ∩ INT(q) = INT(q) ∩ INT(p),* and ditto for positions.

- Proposition. The operations ^ and ˅ are associative.
- For positions: *([p]^[q])^[r] = [q]^([p]^[r])* and: *([p]˅[q]) = [q]˅([p]˅[r])*
- For points: *(p^q)^r = p^(q^r)* and: *(p˅q)˅r = p˅(q˅r)*
- Proof. Set intersection is associative so: *(INT(p) ∩ INT(q)) ∩ INT(r) = INT(p) ∩ (INT(q) ∩ INT(r)),* and ditto for positions.

Hence we can unambiguously write: *p^q^r^…^t* and: *p˅q˅r˅… ˅t,* and similar for positions. We can use single operators operating on sets of points (analogous to *Σ* instead of + in arithmetic), thus:

- Definition. *^(p,q,..r) ≡ p^q^…^r* and ditto for positions.

- Definition. $^\vee(p,q,..r) \equiv p^\vee q^\vee ... ^\vee r$ and ditto for positions.

Note the operations are also distributive for *positions*, with ^ distributing over $^\vee$.

- Proposition. Distributive law for positions. $([p]^\vee[q])\text{^}[r] = ([p]^\vee[r])\text{^}([q]^\vee[r])$.

However the distributive law does not hold for points. Generally: $(p^\vee q)\text{^}r \neq (p^\vee r)\text{^}(q^\vee r)$. They are equal when either: $(p^\vee q)$, $(p^\vee r)$ and $(q^\vee r)$ all exist, e.g. in the A-lattice, or when all are null.

General *Diamond* property.

There is a general solution for finding the *position interior and exterior intersections*: $\text{^}([p],[q],...[r])$ and: $^\vee([p],[q],..[r])$. With coordinates: $[p] = [x_p,y_p]$, etc, we define the four quantities:

- $x_- = X_{min}([p],[q],...[r]) = min(x_p, x_q,... x_r)$
- $x_+ = X_{max}([p],[q],...[r]) = max(x_p, x_q,... x_r)$
- $y_- = Y_{min}([p],[q],...[r]) = min(y_p, y_q,... y_r)$
- $y_+ = Y_{max}([p],[q],...[r]) = max(y_p, y_q,... y_r)$

Then we can write the general solution:

- Proposition.
- $\text{^}([p],[q],..[r]) = [x_-,y_-]$ and: $^\vee([p],[q],..[r]) = [x_+,y_+]$

And similarly we can define the *left* and *right* positions:

- $<([p],[q],..[r]) = [x_-,y_+]$ and: $>([p],[q],..[r]) = [x_+,y_-]$

Geometrically, we are just defining the corners of a rectangle on the position lattice that contains all the points.

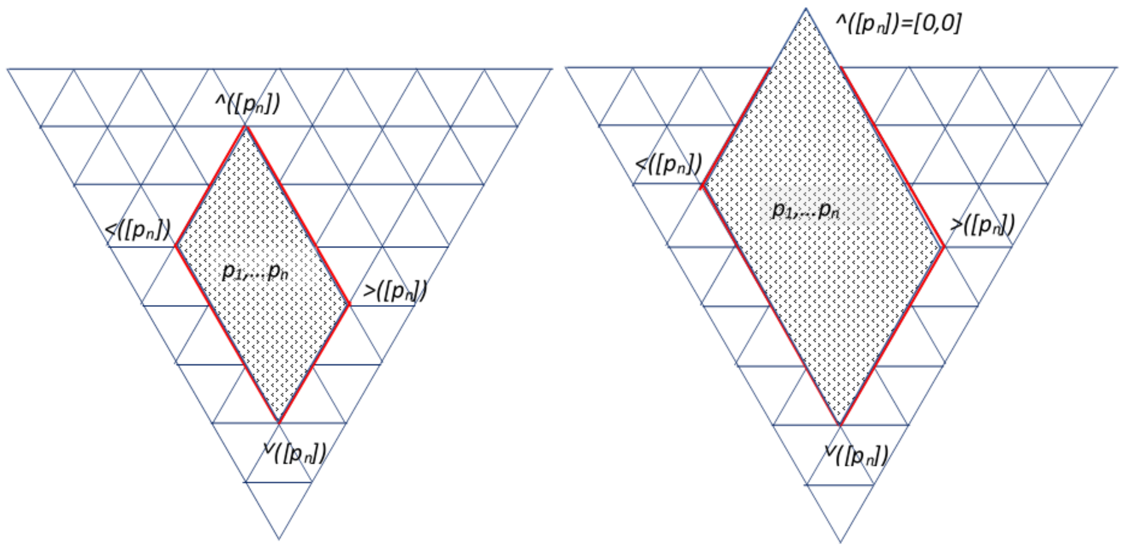


Figure 2.13. A set of positions *{p$_n$}* defines a rectangle enclosing the positions. Left diagram: the positions are enclosed within the main lattice. Right diagram: the positions are too wide, and the top position is *[0,0],* i.e. the CP.

A similar solution for interior and exterior intersections of *points* is not possible however, because two points at specified positions may be joined by points anywhere from *[p]^[q]* down to the CP. We have seen examples previously. So there is not necessarily an interior intersection at: *p^q.* And if there is, there are not necessarily any exterior points at *p*$^{\vee}$*q*.

To calculate the intersection above we must examine the interior and exterior sets, *INT(p)* and *EXT(p).* The *Diamond property* tells us *where to look for the joins (joins and meets).* Instead of recursively comparing all points in the exteriors of *p,q* to find intersections, which becomes unscaleable in large datasets, we need only compare points at specific positions. We check first if *p^q* exists, and then look for shared points at position: *[p]*$^{\vee}$*[q].* The exterior intersection: *EXT(p)∩EXT(q)* is the *exterior of the points: EXT(p*$^{\vee}$*q).*

This also means there is a generalised version of the *Diamond property*, viz.

- *Diamond Property* Generalised. If a class of points: *(p,q…r)* in a FCAT relation have any shared point in their exteriors, they must have a unique shared point in their interiors, *^(p,q,…r*) at the unique position: *^([p],[q],…[r]).*
- Formally. If: $^{\vee}$*(p,q,…r)* is not empty, there is a single point: *^(p,q,…r)* at position: *^([p],[q],..[r]).*
- Proof by induction. (1) It holds for *p,q* by the original *Diamond property*. (2) Induction step. Assume this holds for *p,q,…r.* Then show it holds for: *p,q,…r,s.* Observe that: $^{\vee}$*(p,q,…r,s)* is either null or contains at least one point. If it is null, $^{\vee}$*(p,q,…r,s)* is null, and the property is

trivially satisfied. Otherwise $^{\vee}(p,q,\ldots r,s)$ contains at least one point, so $^{\vee}(p,q,\ldots r)$ contains at least one point. The position: $[t]$ of every point: $t \varepsilon\, ^{\vee}(p,q,\ldots r)$ is the same: $^{\vee}([p],[q],..[r]) = [t]$, and: $\wedge([p],[q],..[r],[s]) = [t]\wedge[s]$. Then by assumption: $^{\vee}(p,q,\ldots r,s) = s^{\vee}t$, and by the simple *Diamond property* (applied to pairs), there is a single lowest point: $\wedge(t,s) = \wedge(p,q,\ldots r)$ at position: $\wedge([p],[q],..[r],[s])$, this is unique, and the property is satisfied. (3) By induction it holds for all finite sets.

Geometric Concepts.

Various CAT2 classes, such as *INT(p), INT(EXT(p)),* etc, may be visualised as manifolds bounded within the larger relation, c_2, and we may define special features, such as *tips, surfaces, volumes, diamonds, lattices, etc,* with analogies to 3-D geometry. These concepts may be regarded as *shapes and manifolds.* In 3-D Euclidean geometry, there is a strict distinction between different *dimensional manifolds,* e.g. *lines (1-D), surfaces (2-D), volumes (3-D), etc.* These have different definitions as class entities, corresponding to (infinite) *1-tuples, 2-tuples, 3-tuples,* etc. Incidentally this makes it very problematic to ever *change the manifold dimensionality* in a continuous multi-dimensional realm. But in CAT2, because the domain is finite, all geometric shapes, including lines, surfaces, volumes, manifolds, have the same type: they are finite subsets of points. They can morph into each other (e.g. by adding or removing or moving joins). This is a big difference from continuous geometry. We also have an interpolated structure of *A-dimensional lattices double joined to a CP within another A'-lattice within another CP', …,* up to the Zero point. This lets us store finite structures recursively within others. These are like trees of manifolds of different finite dimension.

However it is the manifold structure within the specific CP that is most important. This is structured principally by *tips.*

- Definition. Tips of a relation. The *tips* of any relation c_2 is a subset: $TIPS(c_2)$ of points that are not left or right parents of any other points in c_2.
- Definition. Tips in a relation. A point p is a *tip:* $p \varepsilon TIPS(c_2)$ in c_2 just in case $EXT(p) = \{p\}$ in c_2.
- Note the *EXT* function is relative to a relation, c_2. $EXT(p) \equiv EXT(p) \cap c_2 \equiv EXT(p,c_2)$.
- I.e. $EXT(p)$ is really a 2-place function: $EXT(p,c_2)$.

We may also define the concept of tips in a point-subset R relative to a C2 relation.

- Definition. Tips in a subset R of a c_2. A point p is a *tip* in a subset R of a C2 relation c_2 just in case: $EXT(p)\cap R = \{p\}$. Note again $EXT(p)$ is defined relative to c_2.

- Note *EXT(p)* is defined on the C2 relation c_2, so *R* may not contain points with *p* as a parent, but may contain other exterior points of *p* in c_2.
- Hence we may have the case where *p is not a tip of R in* c_2, although *p* is a tip in *R.*

The class of *tips* is defined for any class of points, not just for C2 or CAT2 relations. E.g. an exterior class *EXT(p)* is not C2, but has a class of tips (which is the same as the tips of: *INT(EXT(p)).* Some propositions are stated as follows. Proofs may be given as exercises.

- The tips of (a set of points) *R* in c_2 are the tips of *INT(R)* in c_2.
- The interior of the *tips* of c_2 is equal c_2. *INT(TIPS(*c_2*) =* c_2*.*
- When a new point is added to an existing relation, it is initially a *tip.*
- The set of tips of a collection of points: *TIPS(p,q,…r)* is the minimal subset: *(s,t,…u)* of *(p,q,…r)* such that: *INT(s,t,…u) = INT(p,q,…r).*
- For *R* a general subset of points: *INT(Tips(R))=INT(R).*
- For *R* a C2 relation: *INT(Tips(R))=INT(R) = R.*

We can define other geometric analogies for C2 relations also. E.g.

- Definition. Surface points.
  - A point is a *right surface point* in c_2 if it has at least one left child point but no right child points in c_2.
  - Ditto for a left surface point.
  - A point is a *surface point* if it is either a right surface point or a left surface point.
- Generalise. Surface points of subsets or manifolds.
  - A point *p* is a *right surface point of a subset R* of c_2 just in case *p* is in *R* and *R* has at least one left exterior point of *p* but no right exterior point of *p* in c_2.
  - *R* may be called a *manifold* of c_2 if *R* is a *subset of* c_2 and *R* is *connected*. This means that for every pair of points *p,q* in *R,* whenever *q* is in the interior of *p*, each chain: *<p,…,q>* in c_2 is in *R.*
- Ditto for a left surface point.
- Definition. Interior surface. A point *p* is in the *interior surface* of *INT(EXT(p,q,r,…))* if it has only one parent in *EXT(p,q,r,…)* and the other parent in *INT(EXT(p,q,r,…)).*
- Definition. Plane. A plane may be an *x-plane* or a *y-plane.* The points in an *x-plane* belong to all possible positions of *x* for a fixed value, Y_0*,* of *y:* $\{[x,Y_0]\}$*.* And conversely for a *y-plane.*
  - E.g. with *A = 5,* the *x*-plane at *y=5 is: {[1,5],[2,5],[3,5],[4,5],[5,5]}.* This includes all the positions along the vector from *[1,5]* → *[5,5].*

- Planes at $y = n$ have n positions, e.g. the plane at $y=3$ is: $\{[3,3],[4,3],[5,3]\}$. (Since the condition is: $A < x+y \leq 2A$).
- The class of all positions in *A* is given by the union of all the *x-planes,* or of all the *y-planes.*
- A specific position may be defined by the intersection of an *x-plane* and a *y-plane.*

These kinds of geometric concepts may be useful, especially with the coordinate concepts. However from a pure mathematics points of view, the defining characteristic of the structure lies more in its topology than geometry.

CAT2 Topology.

The most important classes in CAT2 (or C2 generally) are the *interiors* and *their unions.* This family of classes provides the open sets to define a natural *topology*, $\mathcal{T}$, on any C2 or CAT2 relation.

- Definition. The open sets of the *topology,* $\mathcal{T}$, *for* c_2 are (i) the interior sets of each point: *INT(p),* and (ii) the unions of these: *INT(p,q,…r)* for points *p, q, …r,* and (iii) the empty set.

Note this class also contains all subsets of c_2 that are C2.

- The topology *T* for any c_2 is identical to the *class of all the C2 sub-relations in* c_2.

The *base of the topology* corresponds 1-1 to the points:

- The class of interiors of individual points: *{INT(p)}* is the base of the topology.

Key to the topology property is that:

- All intersections of interiors are given by unions of interiors.

This is simply because any intersection: *INT(P)*∩*INT(q)* is the union of interiors of all the (finite number of) *tips* in the intersection, and all finite unions of points interiors are in *T*. We prove it below in terms of tips. We now formally demonstrate the property of the *topology.*

- Definition. Topology on a set of points. [Wikipedia].

Let *X* be a set and let *τ* be a family of subsets of *X*. Then *τ* is called a *topology on X* if:

1. Both the empty set and *X* are elements of *τ*
2. Any union of elements of *τ* is an element of *τ*
3. Any intersection of finitely many elements of *τ* is an element of *τ*

If *τ* is a topology on *X*, then the pair (*X*, *τ*) is called a *topological space*.

[WIKIPEDIA. General Topology.]

We now show that any C2 relation c_2 is a topological space: *(X,T)*, on the set of points *X* = c_2, with the family $\mathcal{T}$ of subsets of *X* including *all unions of interior sets, and the empty set*.

- Definition. The C2 Topology. Define: $\mathcal{T}$ = *{INT(p,q…,r)}* ∪ *{φ},* for any: *p,q,…,r* in relation c_2.
- Proof this is a topology. See definition above.
  1. *INT(EXT(0))* = c_2 is in *T*. *φ (empty set)* is in *T.*
  2. Union of any two elements of *T*: *INT(p,q…,r)* ∪ *INT(s,t…,u) = INT(p,q…,r,s,t,…u)* which is an element of *T.* (By associativity of set unions.)
  3. Consider the intersection: *INT(p)* ∩ *INT(q).* There is a finite set of points: *(r,s,…t)* in the intersection. The interior union: *INT(r,s,…t)* is in *T.* The same argument applies for the intersection of any number of elements of *T*. Hence any intersection of finitely many elements of *T* is an element of *T.*

Note we may also visualise intersections in terms of *tips.* Take the class of *tips* of the intersection: $\{t_\nu\}$ *= tips(INT(p)* ∩ *INT(q))*. The union of interiors of the tips: *INT(*$\{t_\nu\}$*)* is the intersection: *INT(p)* ∩ *INT(q)*. The interior of each tip t_ν is in *T.* Hence the union of the elements: *INT(*$\{t_\nu\}$*)* is in *T.*

Union and Intersection Classes.

We write *interior unions* and *intersections* as follows, using "," for union and "|" for intersection:

- *INT(p,q,…,r) ≡ INT(p)* ∪ *INT(q)* ∪ …∪ *INT(r))*
- *INT(p|q|…|r) ≡ INT(p)* ∩ *INT(q)* ∩ … ∩*INT(r))*

And combinations of set operations in the parameter string, using "," for union and "|" for intersection, e.g.:

- *INT(p,q|r) ≡ (INT(p)* ∪ *INT(q))* ∩ *INT(r)*
- *INT(p|q,r) ≡ (INT(p)* ∩ *INT(q))* ∪ *INT(r)*
- *INT(p,(q|r)) ≡ (INT(p)* ∪ *(INT(q)* ∩ *INT(r))*

We can bracket terms, e.g.: (*p,(q|r)), etc,* to symbolise all Boolean set-theoretic operations.

This symbolism works equally for exterior set unions and intersections:

- *EXT(p,q,…,r) = EXT(p)* ∪ *EXT(q)* ∪ … ∪ *EXT(r)),*
- *EXT(p|q|…|r) = EXT(p)* ∩ *EXT(q)* ∩ … ∩*EXT(r)),*
- etc.

Remember *exterior sets* are not C2 (or CAT2). We must take interiors of exterior sets to get C2s.

- Each point *p* of c_2 maps to a unique element: *INT(p)* of *T.*
- Each point *p* of c_2 maps to a unique subset: *EXT(p)* of c_2.
- Each point *p* of c_2 maps to a class: *INT(EXT(p))* which is an element of *T*.

Note that each *exterior* is unique to its points. If: *EXT(p) = EXT(q)* for two points, then *p* and *q* are in each other's exteriors, and *p=q.* However note:

- *INT(EXT(p))* is not necessarily unique.
- There may be two points, *p≠q,* with the same: *INT(EXT(p)) = INT(EXT(q)).*
- *INT(EXT(p)) = INT(EXT(q))* if:
    - *q* is the *sole child* of *p.*
    - *r* is the sole child of *p* and *q.*
    - *q is the sole child of r which is the sole child of p.* Etc.
- More generally: *INT(EXT(p)) = INT(EXT(q))* iff *p* and *q* are connected by a path of singular points in an exterior lattice.
- *INT(EXT(p)) = INT(EXT(q))* defines an equivalence relation.

The *INT(EXT(p))* classes are in *T*.

- Each class: *INT(EXT(p))* is in *T.*
- If *p* is a tip, *INT(EXT(p)) = INT(p).*
- If *p* is not a tip, *INT(EXT(p)) > INT(p).*
- *INT(EXT(p)) = INT(tips(EXT(p)))*

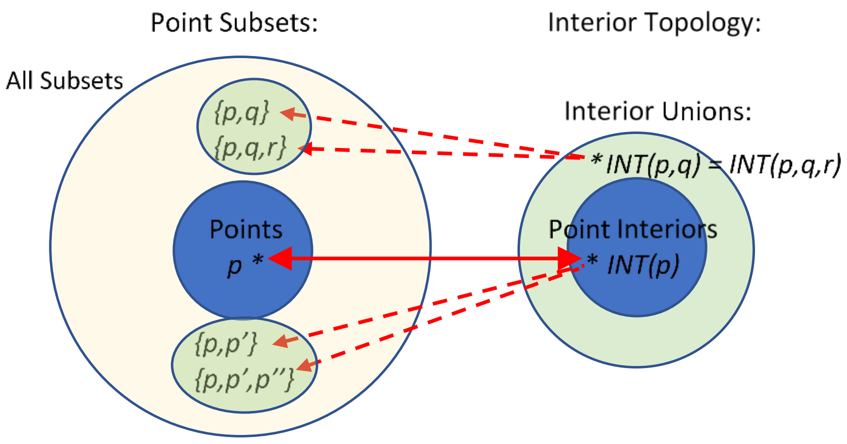


Figure 2.14. The class of points (left) corresponds 1-1 with the point-interiors (right). Unions of point interiors, right, correspond *1-many* to subsets of points, left. The interior topology induces classes on the (Boolean lattice of) subsets of points. It is coarser than the set topology.

The INTEXT Class.

Two important classes are the *interior of exterior intersections,* and the *interior of exterior unions,* which we write as the *INTEXT* function.

- Definition. *INTEXT(p|q|…|r)* = *INT(EXT(p|q|…|r))*
  = *INT(EXT(p) ∩ EXT(q) ∩ … ∩ EXT(r))*
- Definition. *INTEXT(p,q,…,r)* = *INT(EXT(p,q,…,r)*
  = *INT(EXT(p) U EXT(q) U … U EXT(r)).*

Note the identity:

- *INTEXT(p|q|…|r) = INTEXT(p˅q˅…˅r)* since: *EXT(p|q|…|r) = EXT(p˅q˅…˅r)*

Note all C2 interior classes are C2, including unions and intersections of interiors.

The key feature is the natural partition of the C2 or CAT2 relation by the sets: *INTEXT(p)*. These are partially overlapping, as points always have partially shared interiors: *INT(p) ∩ INT(q) = INT(p|q)* and may have partially shared exteriors: *EXT(p) ∩ EXT(q) = EXT(p|q) = EXT(p˅q)*.

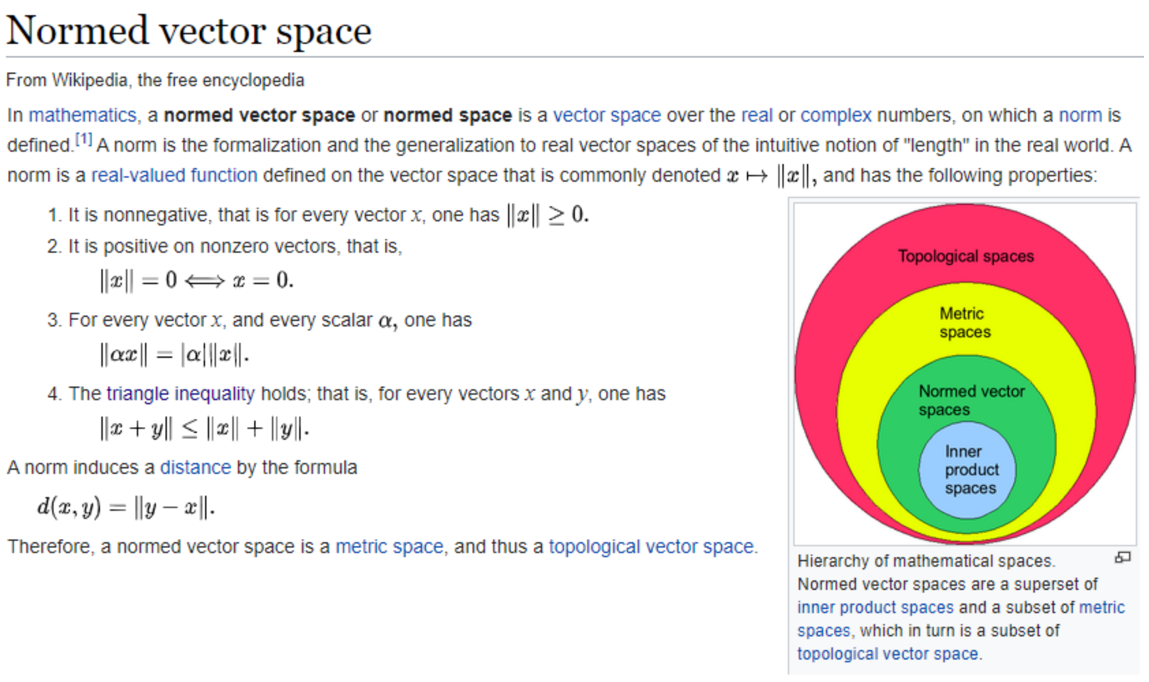
- *INT(EXT(p))* is the smallest CAT2 relation representing all points directly related to *p.*
- *INT(EXT(p))* includes all points *q directly related to p,* and all interior points of those points.
- Points *p* and *q* are said to be *directly related* if they are in a *chain: <p,…,q> or: <q,…,p>,* or equivalently, if either *q is in the interior of p,* or *q is in the exterior of p.*
- Note: *<p,…,q>* iff *q is in the interior of p.* And: *<q,…,p>* iff *q is in the exterior of p.*

For the database application it is important to be able to *partition* the CAT2 (or CAT4) table effectively, to be able to work with partial subsets of records of relevance at a time. The CAT4 database relation is able to grow indefinitely large as we accumulate records, but it is naturally partitioned into separate, semi-autonomous datasets. The complete subset of a CAT2 relation directly related to (at least one of) several points: *p,q, … ,r,* is represented by the union: *INTEXT(p,q,…r).* The complete subset relevant to all these points at once is represented by the intersection: *INTEXT(p|q|…|r).*

The ability to generate the sets is critical for *scalability over a large network.* Instead of working with all the data at once, we can create subsets relevant to subjects of interest. Note there is no comparable ‘partitioning’ operation w.r.t *content* in relational databases, where a partition of all the information on a specific *subject* (insofar as it is defined) depends on partitioning record-sets from many tables, and depends on complex relationships and special ‘business rules’ that may be encoded in functions and programs (information outside the soft-data tables and SQL table joins). See Part 3.

Metric Space and Distance.

C2 (and thus CAT2) has a natural *distance function between points*, viz. the length of the shortest path/s to traverse the graph between the points. We analyse the CAT2 coordinate system as a *metric space.* The distance between *positions* is a simple metric, but does not represent the distance concept between *points.* A metric concept of distance between points also applies.


Normed vector space

From Wikipedia, the free encyclopedia

In mathematics, a **normed vector space** or **normed space** is a vector space over the real or complex numbers, on which a norm is defined.[1] A norm is the formalization and the generalization to real vector spaces of the intuitive notion of "length" in the real world. A norm is a real-valued function defined on the vector space that is commonly denoted $x \mapsto \|x\|$, and has the following properties:

1. It is nonnegative, that is for every vector x, one has $\|x\| \geq 0.$
2. It is positive on nonzero vectors, that is,
   $$\|x\| = 0 \iff x = 0.$$
3. For every vector x, and every scalar α, one has
   $$\|\alpha x\| = |\alpha|\|x\|.$$
4. The triangle inequality holds; that is, for every vectors x and y, one has
   $$\|x + y\| \leq \|x\| + \|y\|.$$

A norm induces a distance by the formula
$$d(x, y) = \|y - x\|.$$

Therefore, a normed vector space is a metric space, and thus a topological vector space.

Hierarchy of mathematical spaces. Normed vector spaces are a superset of inner product spaces and a subset of metric spaces, which in turn is a subset of topological vector space.

[Wikipedia. Normed Metric Space.]

We first analyse this for the simple case of the *position vectors.* Define the *norm* (length) of vectors as before as:

- $|\boldsymbol{v}| = |x|+|y|$

where the vector is: $\boldsymbol{v} = [x,y]$, and $|x|,|y|$ are the absolute values of x,y. The conditions (above) for a normed metric space are satisfied as follows.

- The norm is non-negative, zero on zero vectors, positive on non-zero vectors.

For addition of two vectors:

- $|\boldsymbol{v_1} + \boldsymbol{v_2}| = |[x_1,y_1]+[x_2,y_2]| = |[x_1+x_2,y_1+y_2]| = |x_1+x_2|+|y_1+y_2|$

Since: $|\boldsymbol{v_1}| + |\boldsymbol{v_2}| = |x_1|+|x_2|+|y_1|+|y_2|$ and: $|x_1|+|x_2|\geq|x_1+x_2|$ and: $|y_1|+|y_2|\geq|y_1+y_2|$, it follows that:

- Proposition. Triangle Inequality. $|\boldsymbol{v_1}| + |\boldsymbol{v_2}| \geq |\boldsymbol{v_1} + \boldsymbol{v_2}|$

And we can define a scalar product for α any integer:

- $\alpha\mathbf{v} = [\alpha x,\alpha y]$

Hence:

- $|\alpha\mathbf{v}| = |[\alpha x,\alpha y]| = |\alpha x|+|\alpha y| = |\alpha|(|x|+|y|) = |\alpha|\,|\mathbf{v}|$

Thus the *position vectors* appear at first sight to provide properties of a *normed metric space.*

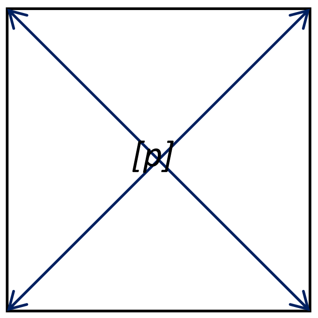


Figure 2.15. Equidistant positions around *[p]* form a square. Since we can only move diagonally, it takes the same path length to reach any points on edges of the square. Thus squares are analogue of Euclidean circles for our position metric.

However this is problematic, first because it only represents vector addition within a *CP,* not across the entire graph, and second because the class of vectors per CP is constrained by the number of atoms, so we do not have: $\alpha\mathbf{v}$ defined as a position vector for all α. The second may be dealt with by distinguishing the *vector space* from the *position space,* as noted earlier. The first requires full vectors across the space, and we will need to incorporate *coordinate chains (across multiple CPs),* and assign norms to these.

However, the most important point is that the norm on *position vectors* represents only the *minimal distance possible* between two points in a CP. It does not represent the natural concept of distance between two *points,* which is *the length of the shortest composite path connecting two points.* To deal with the point metric we need composite chains.

Composite Chains.

To deal with *paths,* we introduce *composite chains,* as our simple chains only go downwards, and can only connect points in the same lattice. We need to represent zig-zagging paths between points in distinct lattices. A *composite chain* is a conjunction of *chains* and *reversed chains.*

- Definition. If *<p,q,…,r>* is a chain, then: *<r,…,q,p>*,* with the points in reversed order, is the *reversed chain.*

While chains go *downwards (to 0),* reversed chains go *upwards*. Note the trivial chain: *<p>* is the only chain equal to its reversal: *<p> = <p>*.*

- Definition. Composite and Simple Chains.
  1. Every chain and reversed chain is a simple chain.
  2. The conjunction of two chains: *<p,…,r><r,…,q> ≡ <p,…,r,…,q>* and similarly two reversed chains: *<p,…,r>*<r,…,q>* ≡ <p,…,r,…,q>*,* where the end-point of the first is the start-point of the second, is a simple chain.
  3. The conjunction of a chain and a reversed chain: *<p,…,r>*<r,…,q>* or: *<p,…,r><r,…,q>*,* where the end-point of the first is the start-point of the second, is a *composite chain*. This is a *chain pair.*
  4. The conjunction of a composite chain and any chain where the end-point of the first is the start-point of the second, is a *composite chain*. E.g. *<p,…,r><r,…,q>*<q,…,s>* or: *<p,…,r><r,…,q>*<q,…,s>*, etc.*
- Composite chains always reduce to alternating chains and reverse chains. This because any sequence of two chains: *<p,…,r><r,…,q>* reduces to one chain: *<p,…,r,…,q>*. and likewise with reversed chains.
- E.g. *<p,…,r><r,…,q>*<q,…,s>** reduces to: *<p,…,r><r,…,q,…,s>*.*
- Alternating chains and reversed chains like: *<p,…,r><r,…,q>*<q,…,s>* cannot be reduced.

We define *path-connections through chains*:

- Definition. *p* and *q* are path-connected by any (simple or composite) chain having *p* and *q* as the start-point and end-point.

The *length of a chain* is defined:

- Definition. Length of a chain.

- A chain or reverse chain with *n* points: *<p_1,p_2,…p_n>* has length: |*<p_1,p_2,…p_n>*| = *n-1.*
- A composite chain: *<p_1,p_2,…p_n><p_n,q_2,…q_m>** has length:

  |*<p_1,p_2,…p_n><p_n,q_2,…q_m>**| = |*<p_1,p_2,…p_n>*||*<p_n,q_2,…q_m>**| = *(n-1)+(m-1)*

- Proposition. The length of any composite chain is the sum of lengths of its simple chains.

Note a pair of chains connecting *p* and *q* may be:

i.	*<p,…,r><r,…,q>*	Single interior chain: *<p,…,q>.*
ii.	*<p,…,r>*<r,…,q>**	Single interior chain: *<q,…,p>.*
iii.	*<p,…,r><r,…,q>**	Interior chain pair.
iv.	*<p,…,r>*<r,…,q>*	Exterior chain pair.

In the first two cases, there are direct chains: *<p,…,q>* or *<q,…,p>.*

In the third case, *p* and *q* are connected by *r* in the *interior.* (An *interior chain*).

In the fourth case, *p* and *q* are connected by *r* in the *exterior.* (An *exterior chain).*

- Proposition. All pairs of points are connected by at least one composite pair.
- Proof: *<p,…,0>* and *<q,…,0>* are chains, hence: *<0,…,q>** is a reversed chain, and *p* and *q* are connected by: *<p,…,0><q,…,0>*.*

Note from the Diamond property, we know that in case (iv), when *p* and *q* are connected by a point in the *exterior,* then (iii) they are also connected by a point in the interior.

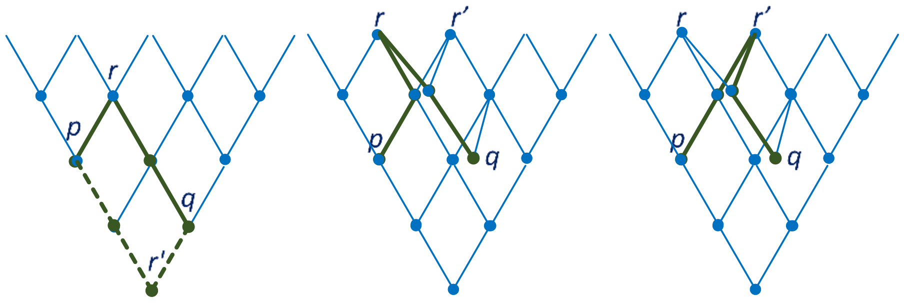


Figure 2.16. Left: *p^q = r* exists and *p* and *q* are connected through *r* by a single interior chain. In this case, *p^{V}q = r'* may exist, and *p* and *q* are also connected through *r'* by an exterior chain. Middle and Left: there is no meet point: *p^q* and no exterior points: *p^{V}q*. In this case, *p* and *q* are connected though (four distinct) shortest interior chains: *<p,.,r,.,q>* and: *<p,.,r',.,q>,* going through points *r, r'* in: *GLB(p,q) = {r,r'}.*

Shortest Chains.

Some further propositions are:

- Propositions. Shortest composite-pair chains.
- There is always *at least one shortest chain* connecting *p* and *q* are through the *interior*.
- Every shortest chain in the interior always goes through a point in *GLB(p,q)* .
- This can always be written as a composite chain: *<p,…,r><r,…,q>** or: *<q,…,r><r,…,p>*.*
- There is generally more than one shortest interior chain.

To prove this, we first we establish the existence of a chain through a point in *GLB(p,q)*, and subsequently show it is the shortest.

- Proof. Existence of the chain. There is always at least one point in the *GLB(p,q).* If *r* is in the *GLB(p,q),* then: *<p,…,r>* and: *<r,…,q>** are chains*.* Hence: *<p,…,r><r,…,q>** is a (composite) chain connecting *p* and *q*. Note if there is a simple chain: *<p,…,q>* then *q* is the *GLB(p,q)*, and: *<p,…,q><q>** is a (composite) chain connecting *p* and *q*. Likewise, if there is a simple chain: *<q,…,p>* then *p* is the GLB, and: *<q,…,p><p>** is a (composite) chain connecting *p* and *q*.

To prove this is the shortest chain we first observe that:

- Proposition. The length of any interior chain: |*<p,…,q>*| is equal to the difference in levels of the start-point and end-point: |*<p,…,q>*| *= L(p)-L(q).*
- Proof. By induction. (i) The length of chain: |*<p>*| *= 0 = L(p)-L(p).* (ii) Suppose the length of a chain: |*<p,…,q>*| *= n = L(p)-L(q).* Then the length of a chain: |*<p,…,q,r>*| *= n+1 = L(p)-L(q)+1.* But: *<q,r>* is a chain of length 1, so *r* is a parent of *q,* and the Level: *L(r) = L(q)-1.* Hence: |*<p,…,q,r>*|*= L(p)-L(r).* (iii) By induction, the length of all chains is: |*<p,…,q>*| *= n = L(p)-L(q).*
- Note the converse holds for reversed chain: |*<q,…,p>**|= |*<p,…,q>*| as they have the same number of points, so: |*<q,…,p>**|*= L(p)-L(q),* the difference in levels of the end-point and start-point*.*

From this we conclude that:

- The shortest chain joining *p* to a point in the interior of *q* is a chain from *p* to any point *r* in GLB*(p,q).* This is because the GLB*(p,q)* is by definition at the highest level in the interior intersection, hence the chain length: |*<p,…,r>*| *= L(p)-L(r),* and since *L(r)* is maximal, this is the minimal length from *p* to the intersection: *INT(p,q)*.
- Similarly, the chain length: |*<q,…,r>*| *=* |*<r,…,q>**| *= L(q)-L(r)* is minimal.
- Since every interior chain connecting *p* and *q* must go through at least one point in *INT(p,q)*, the chain length: |*<p,…,r>*|*<r,…,q>**| is minimal.

Hence we can summarise:

- The chain length: |*<p,…,r>*|*<r,…,q>**| = *L(p)+L(q)-2L(r),* where: *r ε GLB(p,q),* is the shortest interior chain length between *p* and *q.*
- Any chain: |*<p,…,r>*|*<r,…,q>**| where *r ε GLB(p,q),* is a shortest chain between *p* and *q.*

In the case where there is an *external intersection,* the Diamond property means this is always at: *p˅q,* and the meet point: *p^q* exists and is the *GLB(p,q)*. We can similarly show that the shortest exterior chain is: |*<p,…,r'>*<r',…,q>*|*,* where *r' ε p˅q.* From the previous coordinate geometry, this is the same as the interior chain length: |*<p,…,r><r,…,q>**|*,* where *r ε p^q.*
Finally we can define the *distance between two points.*

- Definition. The distance: |*p,q*| is the length of the shortest chain connecting *p* and *q.*
- Hence in general: |*p,q*| = *L(p)+L(q)-2L(r),* where: *r ε GLB(p,q).*
- E.g. if *q* is in the interior of *p,* then: *p^q = q,* and: |*p,q*| = *L(p)+L(q)-2L(r) = L(p)-L(q).*

We can now see this distance function defines a *metric,* representing distance between two points.

The key property is that the *triangle inequality holds.* i.e.

- Proposition. Triangle inequality. The distance: |*p,q*|+|*q,r*| ≥ |*p,r*|.
- Proof. If there are two shortest chains: *<p,…,q>* and: *<q,…r>* from *p* to *q* and *q* to *r* respectively, there is a chain: *<p,…,q,…r>* from *p* to *r.* Then: |*<p,…,q,…r>*| = |*p,q*|+|*q,r*|. If: *<p,…,q,…r>* is a shortest chain from *p* to *r,* then: |*<p,…,q,…r>*| = |*p,r*| = |*p,q*| +|*q,r*|. If there is a shorter chain: *<p,…r>* then: |*<p,…r>*|<|*<p,…,q,…r>*|, so: |*<p,r>*|<|*p,q*| +|*q,r*|. In either case the triangle inequality holds.

This is because given: *<p,…,q>* and: *<q,…r>,* there is always at least one path: *<p,…,q,…r>* (we can go from *p* to *q* then *q* to *r*), with distance: |*p,q*|+|*q,r*|, giving the maximal bound for: |*p,r*|. Hence the *distance function* between points has the triangle inequality property of a metric.
Note shortest paths do not need to pass through the GLB, but there is always at least one shortest path connecting *p* and *q* through the GLB of *p* and *q*.

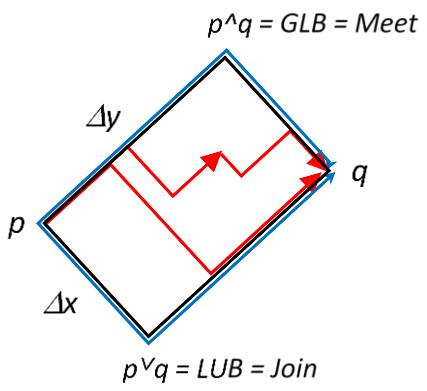


Figure 2.17. Shortest paths from *p* to *q* when there is a LUB. Every shortest path (e.g. red, blue) from *p* to *q* traverses a total of *Δx* steps (right-down) and *Δy* steps (right-up), taking steps in any order. This always reaches *q* in the minimal number of steps, viz. *Δx+Δy.* The two blue paths are also shortest paths, and are the only paths to include the GLB and LUB.

The number of shortest paths between *p* and *q in the interior of a specific point in p˅q* is the number of ways to place *Δx* objects in *Δx+Δy* positions, divided by the permutations *Δx!* which is:

- *Number of shortest paths = (Δx+Δy)!/Δx!Δy! = C(Δx,Δy).*

Remember this metric (distance between *p* and *q*) only holds when there is a GLB. To reiterate:

- Proposition. If *p,q* has a *LUB,* it is at the join position: *[p]˅[q],* and the GLB is the meet point: *p^q* (Diamond property).
- Then there is one shortest chain each through the *LUB* and the *GLB* respectively, but many others as well.

The feature that there may be many shortest paths between two points is unlike 2-D or 3-D Euclidean geometry.

We might now ask if this corresponds to a *normed vector space, inner product space,* or some other special class of spaces. But we leave this analytic focus now, and move on to describe some other features of the CAT2 structure directly. Next we consider the GLB intersection in the case where there is no *p^q* point.

GLB and Intersection Matrix.

For any two points *p* and *q* in the same CP, the GLB of *p,q* may occur at any level, down to the CP in the interior of the position intersection: *INT([p]^[q]).*
The GLB intersections have a relatively simple algebraic pattern. The interiors of *p* and *q* act exactly like 2-D intersecting triangular sheets of 'mesh' that are joined along the *atomic row,* but separate at certain points, up to the *[p]^[q]* position of *p* and *q*.

- Proposition. Points in the union: *INT(p,q) = INT(p)* U *INT(q)* at any position *[x,y] across any constant Level: L = x+y-A*, i.e. where *x + y = Constant,* may or may not be in the intersection: INT(*p|q),* independently of intersections at any other positions in the same Level.
- Before giving a proof, we add some useful functions. We can write *classes at positions* as functions using the *".[x,y]"* notation*.*
    - *INT(p,q).[x,y] = {r* ε *INT(p,q) & [r] = [x,y]}*
    - *INT(p|q).[x,y] = {r* ε *INT(p|q) & [r] = [x,y]}*
- We now construct a binary sequence: ***$(\alpha_1,\alpha_2,...\alpha_L)$*** where: ***$\alpha_l$* = 1 or 2, t**o represent the counts of the *union: INT(p,q).[i,L-i+1],* across a level: *x+y = L.*
- Note the position-intersection is singular, the union is binary.
    - The union: INT(p,q).[x,y] has two points if: *INT(p|q).[x,y]* is null.
    - The union *INT(p,q)* has one point if *INT(p|q).[x,y]* has one point.
- Use sequences of ***{1,2}*** values to represent the counts of the interior position-unions: *INT(p,q).[x,y]*.
- For convenience, we may switch to using interior coordinates and interior Levels, taken from the meet position: *[p]^[q].* We may set this as *[0,0]* at Level 1*,* with coordinates *[x,y]* increasing to the *atomic level* at Level *A,* where: *x+y=A-1.* We now take levels: *L = x+y+1.*
- This means there is now a sequence of *L* values: ***$(\alpha_1,\alpha_2,...\alpha_L)$*** in the row at each level *L*.
- We may call a matrix of such values, from *L = 1* to *L = A,* the *intersection profile* of the points.
- The proposition is then that every possible sequence of two values, e.g.: ***(2,2,1,2,2,1,1,2)*** of length *L, e.g. 8,* represents a *possible intersection profile* for points *p, q,* at level *L.*
    - Note this holds as long as the level *L* is between their position intersection: *[p]^[q]* and the atomic level, where the row is determined to be singular: ***$(1_1,1_2,...1_A)$***.
    - Note if *INT*(*p)* and *INT(q)* do not intersect in the CP exterior, they intersect at the CP, and this is the only intersection and union point and is a trivial case.
- Proof. This is possible if we can construct a valid CAT2 graph at intermediate levels between *L = 1* and *L = A* that is consistent with the sequence: ***$(\alpha_1,\alpha_2,...\alpha_L)$*.** We now show we can

construct at least one graph. This is the minimal graph, in which every row in the intersection profile above *L* is filled with 2's: ***$(\alpha_1,\alpha_2,\ldots\alpha_{L-i}) = (2_1,2_2,\ldots2_{L-i})$***, and every row below *L* is filled with 1's: ***$(\alpha_1,\alpha_2,\ldots\alpha_{L+j}) = (1_1,1_2,\ldots1_{L+j})$***.

- First construct part of the simple flat A-lattice, from the atomic row to the level before *L,* with a single point in each position. This has count profile: ***$(1_1,1_2,\ldots1_{L+j})$***, at each level from *L+1 to A.*
- Then for each α_i in: ***$(\alpha_1,\alpha_2,\ldots\alpha_L)$***, at Level *L,* insert either one or two points, according as α_i is ***1*** or ***2***, in the interior position: *[i-1,L-i].* In each case there are unique points at the interior lattice positions: *[i-1,L-i+1]* and: *[i,L-i],* providing left and right parents.
- Wherever there are two points in a position in *L*, label one "P" and the other "Q". These are non-intersecting points, and belong to separate lattices: *P ≡ INT(p), Q ≡ INT(q)*.
- Wherever there is one point in a position in *L*, label it "P|Q", (meaning "P and Q"). These are intersection points. And belong to: *P|Q ≡ INT(p|q)*.
- Then for each α_i in: ***$(\alpha_1,\alpha_2,\ldots\alpha_{L-1})$***, at Level *L-1,* insert two points in each position: *[i-1,L-i-1].* In each case, there may be either one or two points available for parents, in both the left and right parent positions, in the previous row *L.*
    - If there is a single point on both sides, they are "P|Q" points. Add two new points, both parenting to the single "P|Q" parents, and arbitrarily label one point "P" and the other "Q".
    - If there are double points on both sides, add one new point parenting to the "P"-"P" pair and label this "P", and add one new point parenting to the "Q-Q" pair and label this "Q".
    - If there is a single and a double parent available, add new points "P" and "Q", joining with parents: "P" $\rightarrow$ ("P","P|Q") and: "Q" $\rightarrow$ ("Q","P|Q").
- Then for each α_i in each subsequent Level: ***$(\alpha_1,\alpha_2,\ldots\alpha_{L-1-j})$***, from Level: *K = L-2 to 1* (or: *j = 1 to L-2),* every position has two points, a "P" and "Q" point. Insert two points in the interior position: *[i-1,K-i],* and in each case use: "P"-"P" pairs and "Q"-"Q" pairs as parents. (The parents are now determined: we can now longer join "P" and "Q" points.)
- When we get to Level 1, or interior position *[0,0],* there are two points, labelled "P" and "Q".
- These belong to two *sheets,* "P" and "Q", that are separate at the edge, but meet at intermediate points, up to the atomic row.

We now observe that the formal pattern of the intersection graph is fully defined by the *intersection matrix.*

- Proposition. The formal pattern of an interior intersection of two points in the same CP is fully represented by the intersection matrix: ***{(α₁,α₂,…α_L)},*** for *i = 1 to A.*
- Any two *interior union graphs*: *INT(p,q),* and: *INT(p',q'),* having the same intersection matrix: ***{(α₁,α₂,…α_L)}*** are isomorphic to each other. (Also vice-versa).
- Hence they represent the same C2 relation, by the Isomorphism Axiom (4).
    - This assumes the *interior union graphs*: *INT(p,q),* and: *INT(p',q')* is taken *up to the CP,* and requires: *[p]=[q]* and: *[p']=[q'].* (This is not really a restriction.)
- Proof. Using the assumptions, we will define a mapping: π*: r* $\rightarrow$ *r'* that maps each point *r* at position *[r]* in *INT(p,q)* to a point *r'* at the same position *[r]* in *INT(p',q'),* and preserves all join relations, hence showing isomorphism.
- First choose: π*(p) = p',* π*(q) = q'.* (It does not matter which order they are paired as they are identical).
- Then take: *INT(p)* and *INT(p').* Each has the same lattice of positions with one point in each position, up to the CP. Hence a 1-1 correspondence.
- First map: *INT(p)* $\rightarrow$ *INT(p').*
    - For each *i* in *INT(p),* define: π*(i) = j'* where *j'* is in *INT(p')* and: *[i]=[j'].*
- Do the same for *q*.
    - For each *i* in *INT(q),* define: π*(i) = j'* where *j'* is in *INT(q')* and: *[i]=[j'].*
- Note this second process will overwrite the first when *i* is in the intersection: *INT(p|q)*, but the mappings do not change.
- This maps all points of *INT(p)* uniquely onto all points of *INT(q)*.
- The mapping π matches the *coordinate structure,* and hence the *C2 join structure*.
- If: *ID1(p) = q,* then: *ID1(p') = q',* because: *ID1(p') = ID1(*π*(p)) =* π*(ID1(p)).*
- Hence there is an isomorphism.

This is quite useful to know. It means every interior intersection of *pairs of points* is fully described by their Intersection matrix.

- Proposition. The number of possible distinct *ordered rows* that can occur at Level *L* in an intersection matric is 2^L.

Thus every logically possible *ordered binary sequence* is a distinct *interior graph row*. However there are strong vertical relations on rows, as illustrated. In the following matrices, using *interior levels,* **2** means *two interior parents for p and q at this position,* **1** means *one interior parent (intersection).* The *atomic row* is taken to be *A* = 10.

1.	**(2)**	← 1 pattern: distinct parents at *[p]^[q]*
2.	**(2,2)**	⬆
3.	**(2,2,2)**	*determined*
4.	**(2,2,2,2)**	
5.	**(2,2,2,2,2)**	⬆
6.	**(2,2,2,2,2,2)**	← 1 possible higher intersection pattern
7.	**(2,2,2,2,2,1,2)**	← 2 possible higher intersections patterns
8.	**(2,2,**1**,2,2,**1,1**,2)**	**← Possible intersection pattern at *L = 8***
9.	**(2,2,**1,1**,2,**1,1,1**,2)**	← 16 possible lower intersections patterns
10.	**(**1,1,1,1,1,1,1,1,1,1**)**	← 1 pattern at *L = A*: single Atomic Parents.

Figure 2.18. A possible intersection pattern for the matrix represents a possible type of CAT2 interior relationship between *p* and *q* where *[p]^[q]* is at Level 10. Here we have fixed the row at level 8 as an arbitrary sequence, and filled the other rows above and below it. Some points, shown in red, have alternative possible values in rows above and below Level 8.

Note the general rule for labelling a possible matrix is simply:

- All values in the atomic row are *1.*
- All cells with *1* must have *1's* in neighbouring cells below.
- All cells with *2* must have *2's* in all neighbouring cells above.

We pose this question:

- What is the number, *N(A),* of distinct intersection matrix patterns possible at given level *A?* Note *N(A)* increases faster than 2^{A-1}, as there are always at least that many possibilities for the *A-1* row, with then between 1 and up to 2^{A-2} subsequent possibilities in the *A-2* row, etc. It must be constrained to at most the exponential quadratic: $N(A) << 2^\wedge(2A)^2$, but most likely to just an exponential: $N(A) < 2^{2A}$.

We now consider another type of approach to classifying CAT2 relations.

Tree Symbols LTS and RTS.

While CAT2 relations are defined in detail by class of points, we want to develop a language for describing relations in a more general structural way. E.g. we have seen we can summarise relations as trees of CP's, characterised by the number of atoms. But this does not capture much detail of structures inside the CPs. We mention here that another kind of description of CAT2 relations may be given, by what we call *C2 Tree Symbols*. The general idea is that this gives a more detailed description of relations without a full point-like specification. In such a system, *types of tree symbols* act as a language to characterise types of sub-relations. They can be also be considered as a kind of type symbols.

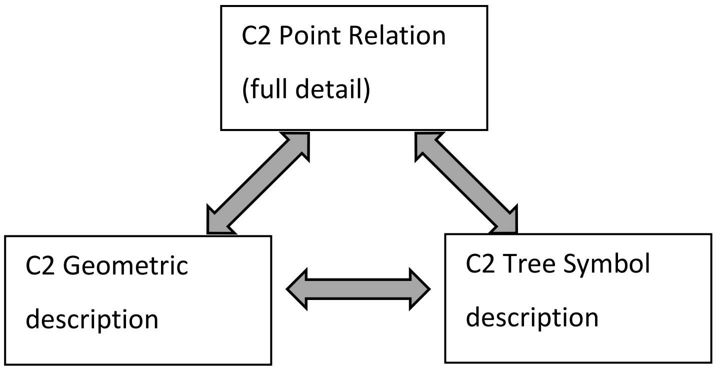


Figure 2.19. Three descriptions at different levels of generality. The translations between these descriptions is of key interest for describing common structures of relations.

Here we introduce the idea of tree symbols and some basic properties.

- Definition. Simple tree symbols for C2. We define a *left* and *right tree symbol* (*LTS, RTS)* for each point, defined recursively as follows. Complete the following process for a C2 (or CAT4) relation to determine the simple tree symbols.
- Step 0. The *LTS* and *RTS* for each point is first labelled "*1".*
- For each point in turn (taken in some fixed order, e.g. the *reversed ID order*) we perform the following three steps to generate a new *LTS* and *RTS* for each point. (The following process executed over all points in turn represents one *update loop,* which is repeated until the symbols are stable.)
  - Step 1. If the point has no right children, leave the *RTS* as "*1".* If it has no left children, leave the *LTS* as "*1".*
  - Step 2 . If the point has right children, label the *RTS* with the string of all the *RSTs* of the right children enclosed in brackets: $(s_1)(s_2)\dots(s_n)$. Ditto for the *LTS*.
  - Step 3. For both symbols. Order the symbols: (s_i) alphabetically in each string: $(s_1)(s_2)\dots(s_n)$, to get identical symbols adjacent. Wherever there are *m* adjacent symbols of the same type: $\dots(s_1)^1(s_1)^2\dots(s_1)^m\dots$, replace them with: $\dots m(s_1)\dots$. (Note this may be done in stages, replacing $\dots n(s),m(s)\dots \rightarrow \dots,(n+m)(s),\dots,$ where *s* is a *symbol type* and *m,n* are whole numbers).
  - If the new *LTS* and *RTS* calculated for a point is the same as the previous one, it is *stable,* and its symbols do not need to be recalculated in the next update loop.

- Repeat the steps *1-3* over the record set until the *left and right tree symbols for all points* are stable.

Examples of replacements.

- E.g. *(1)(1)(1)* → *3(1).*
- E.g. *(2(1))(2(1))(2(1))(3(1))(3(1))* → *3(2(1))2(3(1))*.
- Note you cannot add symbol types through brackets.
- E.g. *…(n(s)),(m^(s))…* cannot be replaced by: *…,(n(s)+m(s)),….*
- E.g. *2(1)3(1)* is replaced by: *5(1)*. But: *(2(1))(3(1))* is not replaced by: *(5(1))*.
- Note when a double bracket term occurs: …,*((s))…* in a string, it may be replaced with: …,*(1(s)),…* e.g.: *((2(1)))* →*(1(2(1))).* However it is not necessary to replace single terms: *(s)* with *1(s).*

The resulting symbols are called *tree symbols,* and symbolise the left and right trees, T_1 and T_2, of C2 exteriors. The symbols accumulate in the interiors, and the zero point symbols represent the complete trees. Note these symbols completely determine the *tree structures.* But do the two tree structures determine the full relation? This is related to finding a method of describing the graph of a C2 relation sufficient to reconstruct it. If there was an efficient means of codifying the C2 relation in a shorter symbolic string, and subsequently reconstructing the relation from this, this would be very useful. Note the tree symbols are much shorter than the list of C2 database records of the 3-point-relations: *(ID, ID1, ID2)*.

Indeed if there was a method of *determining the C2 graphs from tree symbols,* this would amount to a higher-level symbolic language for specifying C2 graphs. Alternatively, it may be that there is no generally effective method to completely specify graphs, short of specifying all the point-relations.

It is quickly found that the simple *tree symbols* do not determine unique reconstructions of graphs. E.g. take the simple relations:

R_1	*A*	*B*
A	*1*	*1*
B	*1*	*1*

R_2	*A*	*B*
a	*2*	
b		*2*

These may be represented as the following CAT2 graphs (see *canonical graph for simple relations,* Part 3).

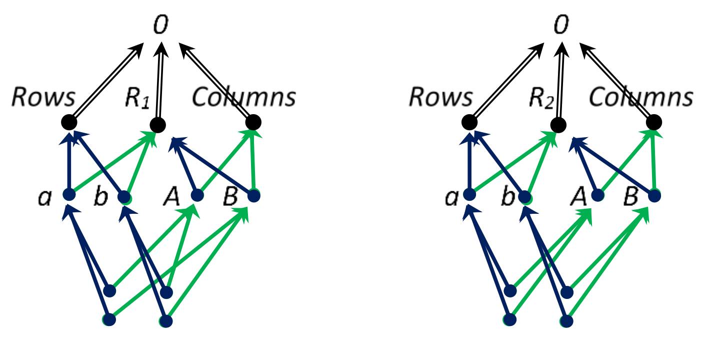


Figure 2.20. Left. Relation R_1. Right. Relation R_2. The relations are distinct, but they have identical tree symbols. The LTS for 0 is: *(2(2(1))(2(1))(1)* and the RTS for 0 is: *(1) (2(1)) (2(2(1)))* in both cases. Note that these are not ordered.

These are the simple tree symbols.

	Relation R_1		*Relation R_2*	
Symbols	*LTS*	*RTS*	*LTS*	*RTS*
ROWS:	*2(2(1))*	*1*	*2(2(1))*	*1*
R:	*2(1)*	*2(1)*	*2(1)*	*2(1)*
COLS:	*1*	*2(2(1))*	*1*	*2(2(1))*
Zero:	*(2(2(1))) (2(1)) (1)*		*(2(2(1))) (2(1)) (1)*	
	(1) (2(1)) (2(2(1)))		*(1) (2(1)) (2(2(1)))*	

Given the tree symbols for the two relations are identical, we could reconstruct either relation. The simple tree symbols provide limited information about how the two trees can be 'spliced' together to reconstitute the relations. A second method gives tree symbols with more information. This is similar to the previous method, but differs in Step 3:

- Step 3. For both symbols.
  - Added Step: Order the *LTS* child symbols: *$(s_1)(s_2)\ldots(s_n)$*, so that symbols for points with the same right-hand-side parents are in adjacent groups, and insert brackets: *{}* around each group, e.g. if there are two right parent groups: *$\{(s_1)(s_2)\ldots(s_i)\}\{(s_j)\ldots(s_n)\}$*. Ditto for *RTS,* bracketing by left-hand-side parents.
  - Then continue as before, ordering the symbols: *(s_i)* alphabetically within each set of curly brackets: *$\{(s_1)(s_2)\ldots(s_n)\}$*, to get identical symbols adjacent. Wherever there are *m* adjacent symbols of the same type: *$\ldots(s_1)^1(s_1)^2\ldots(s_1)^m\ldots$*, replace them with: *$\ldots m(s_1)\ldots$*.

- Then order the bracketed symbols: *{s}{s'}...{s''}* alphabetically to get identical symbols adjacent. Wherever there are *m* adjacent symbols of the same type: $\{s\}^1\{s\}^2\ldots\{s\}^m$, replace them with: $m\{s_1\}$.

This involves a triple ordering of the child symbols: first by their right (left) groups, then within *{}* brackets by symbol types, then between bracketed terms, by symbol types within the brackets.

These symbols give more information about the graph construction. E.g.

Symbols	*LTS*	*RTS*	*LTS*	*RTS*
ROWS:	*{2(2{(1)})}*	*1*	*{2({2(1)})}*	*1*
R:	*{2(1)}*	*{2(1)}*	*{2(1)}*	*{2(1)}*
COLS:	*1*	*{2(2{(1)})}*	*1*	*{2({2(1)})}*
Zero:	*{({2(2{(1)})}) ({2(1)}) (1)}*		*{({2({2(1)})}) ({2(1)}) (1)}*	
	{({2(2{(1)})}) ({2(1)}) (1)}		*{({2({2(1)})}) ({2(1)}) (1)}*	

Notice now the left hand relation has: *{2(2{(1)})}* as *LTS* for the *ROWS* point, while the right hand relation has instead: *{2({2(1)})}*. These symbols now fully determine these two relations.

However these symbols are still not deterministic. The smallest relation they fail to distinguish appears to be *3 X 4,* e.g.

R₁	*A*	*B*	*C*	*D*
A	*1*	*1*		
B	*1*	*1*		
C			*1*	*1*

R₂	*A*	*B*	*C*	*D*
a	*1*	*1*		
b	*1*		*1*	
c		*1*		*1*

Figure 2.21. Two distinct (non-isomorphic) relations that are not distinguished by the enhanced CAT2 tree symbols.

We can indicate the form for these *CAT2 tree symbols* concisely like:

- Definition. CAT2 Tree Symbols.
- 1. "*1*" is a TS.
- 2. If $a_1,a_2,\ldots,a_m$ and $b_1,b_2,\ldots,b_n$ are whole number symbols and $s_{1,1}\ldots, s_{i,j},\ldots, s_{m,n}$ are *TS's,* then $a_j\{b_i(s_{j,i})\}$ is a TS,

Where we define the iterated syntactic operator: $a_j\{b_i(s_{j,i})\}$ ≡ $\&_{j=1 \text{ to } M}\, [a_j\{\&_{i=1 \text{ to } N}\, [b_i(s_{j,i})]\}]$ by primitive syntactic substitution operators, where the symbol (not TS): $\phi(j) = \&_{i=1 \text{ to } N}[b_i(s_{j,i})]$ is the repeated conjunction: $b_1\&"(“\&s_{j,i}\&“)" \;\&\; \ldots \;\&b_n\&"(“\&s_{j,n}\&“)"$, and similarly: $\&_{j=1 \text{ to } M}\, [a_j\{\phi(j)\}]$ is the repeated conjunction: $a_1\&"\{“\&\phi_1\&“\}" \;\&\; \ldots \;\&a_m\&"\{“\&\phi_m\&“\}"$.

This lets us generate a recursive set of symbols, like:

L1	*L2*	*L3*	*L4*
1	*1{1(1)}*	*1{1(1{1(1)})}*	*1{1(1{1(1{1(1)})})}*
	1{1(1)1(1)1(1)}	*1{1(1)1(1)1(1)} = 2{3(1)}*	
	2{3(1)}	*7{8(2{3(1)})}*	*11{12(7{8(2{3(1)})})}*
	4{5(1)}	*9{10(2{3(1)})}*	*etc*
	2{3(1)}4{5(1)}	*2{3(1)}4{5(1)}{6(1)}7{8(2{3(1)})}*	

Figure 2.22. Illustration of tree symbols, in levels. *Tips* have symbol *“1”*. Every pair of *“{}”* of curly brackets represents a *parent point,* in the opposite tree*.* Every pair of *“()”* brackets represents a point in the exterior.

This defines the class of all symbols including *denormalized symbols.*

E.g. these symbols reduce to each other: the most compact is called *fully normalised.*

1{1(1)1(1)1(1)}1{1(1)1(1)1(1)} = 1{3(1)}1{3(1)} = 2{3(1)}

We can define:

- *Fully normalised symbols* are ordered, with all sequences of terms of the form: $a(\phi_1)$ within the curly brackets: $\{\ldots a(\phi_1)b(\phi_2)\ldots\}$ ordered alphabetically so: $\phi_1<\phi_2$, and conversely, with all sequences of terms of the form: $a\{\phi_1\}$ within the round brackets: $(\ldots a\{\phi_1\}b\{\phi_2\}\ldots)$ ordered alphabetically so: $\phi_1<\phi_2$,:
- *Fully normalised symbols* have no repetitions of symbol types ϕ in sequences like: $\{\ldots a(\phi)\ldots b(\phi)\ldots\}$ or like: $(\ldots a\{\phi\}\ldots b\{\phi\}\ldots)$. These reduce to: $\{\ldots c(\phi)\ldots\}$ and: $(\ldots c\{\phi\}\ldots)$ respectively, where *c* is the symbol for: *a+b.*
- *Fully de-normalised symbols* contain only brackets and *“1”’s.*
- *Fully de-normalised symbols* are generated by taking all terms like: $\ldots a(\phi)\ldots$ with $a > 1$, and replacing with: $\ldots 1(\phi)^1 1(\phi)^a\ldots$, and similarly replacing all: $\ldots b\{\phi_1\}\ldots$ with: $\ldots 1(\phi)^1 1(\phi)^b$.
- Proposition. There is a unique fully normalised TS for every CAT2 TS.
- Proposition. There is a unique fully de-normalised TS for every CAT2 TS.
- Proofs. Exercise.

Note the enhanced symbols are still only *tree symbols*, i.e. they correspond uniquely to (pairs of) trees. This is seen by simply replacing curly brackets with round brackets, which gives back original simple tree symbols. For this reason it appears such symbols cannot uniquely correspond to C2 relations, otherwise we would have a general isomorphic mapping between trees and C2s, which is not possible.

- This requires proof.

Perturbations of the network below result in alterations of the collected TS above. But the symbols may change their *numbers.*

- Every point we add is (initially) a *tip.*
- If we add a point in a new position, it will perturb the TS.
- If we add a point in a position with tips, it will perturb numbers in the TS.

E.g. adding another point into: *1{n(1)}* gives: *1{(n+1)(1)}.* But if there were *m* of these: *{(m{n(1)})}* then introducing an extra point changes it to: *{((m-1){n(1)})({(n+1)(1)})}.* This propagates all the way to the CP. When a new point is inserted within a symbol effects in the resulting string . We can take the interior of the point, and *de-normalise* the TS representations w.r.t. points in this interior.

By contrast, the *relational symbol, RS,* may be put as a string like this:

“$\{(0,0,0),(1,j_1,k_1),\ldots (\nu,j_\nu,k_\nu),\ldots (i_N,j_N,k_N)\}$”

Where: j_1,k_1 and ν, j_ν, k_ν , are replaced by number-symbols, ν is unique, and *j, k* may be numerals up to ν. This string takes about: 3N long-integer symbols. This means it is ordered by the first variable.

If we store this in adjacent records, then we can quickly find subsets of records queried by values in the *(i,j,k)* positions. Time to find *i* is constant with *N.* (*Skip 3i*). Time to find *j’s* can be made like *Log(N).* So this is scalable.

A further enhancement to the tree symbol is to impose an *order condition.* It is possible to require ordering of symbols for *CP atoms* matching in some respects between the LTS and RTS. They are then no longer purely *tree symbols.* But this may be used to match the two symbols more closely when it comes to recombine them.

There is an analogy here with the double-stranded DNA, and the splitting and recombination process of the chromosome. Two half-strands of DNA are combined, and they reproduce a cell, without being fully deterministic as to the outcome, and with certain potential mutations. Similarly, the LTS and RTS from one or separate CAT2 tables may be recombined, with indeterministic results.

Conditions for ambiguity of relations relative to the tree symbols are interesting. With the simple tree symbols, any two relations with the same *column* and *row sums* have identical tree symbols. With the enhanced tree symbols, only two relations with the same *column* and *row sums* and the *same numbers* have identical symbols. But there can be different distributions of the same numbers in the table cells corresponding to distinct, i.e. non-isomorphic, C2 relations, as in the example above. There are relations which do have unique tree symbols, and these can be regarded as 'primes'. This leads to questions about the ability to codify C2 relations in such strings. Also, this leads to methods of *normalising* relations so they will not be ambiguous. This is an extensive topic, and we now move on.

Combinatoric problems.

Before going on with the interpretation, we briefly mention problems of combinatorics or group theory, counting types of C2 or CAT2 relations, etc. Basic questions are such as:

- How many possible C2 relations are there with *N* points? Ditto CA2, CAT2, FCAT2.
- How many possible C2 atomic sequences are there of length *A* from *D* atoms?
- How many CAT2 or FCAT2 relations are there with *A* atoms and *N* points?
- How many points are there in an interior lattice with *A* atoms? (Answer: *A(A-1)/2*)

Combinatoric problems can be difficult, and we do not need go into these here, but it is useful to have an idea of how the cardinalities of the classes are related, especially for some questions of scalability. E.g. for a free directed bi-graph, where any point (vertex) can have a single join type (edge) to any other, the number of possible graphs where points are distinct is: $(2^N)^N$. (With non-distinct points we can divide by *N!*) This is a very explosive function. If there is just one join for every point it is: N^N, which is much smaller, but still explosive. This is why the space of free graphs is too large to be effective to work with, and imposing structure is essential. The number of C2's is of much smaller order, and CA2, CAT2, FCAT2 are smaller again.

Using *C* as a constant and *N* as integer variable, we may rank recursive functions types, e.g. like:

- C *(constant)* $< \log(N) < N < N^C$ *(polynomial)* $< c^N$ *(exponential)* $< N!$ *(factorial)* $< N^{CN} < (c^N)^N <$ $(\ldots(N^\wedge N_1)^\wedge N_2)\ldots)^\wedge N_c$, etc. (Goes on forever).

Each function type eventually overtakes the previous one, for fixed constant *C,* as *N* increases.[4] But just to orient ourselves to the cardinalities of the C2 classes:

- Class of all unary relations with *N points* $\rightarrow N^N$
- (We think) Class of CAT2 relations with *N* points $\rightarrow$ *N!* (Needs proof.)

The first represents the 'logical space' of which C2 is a subspace. The FCAT lattices may be similarly compared to atomic Boolean lattices:

- Points in atomic Boolean lattice $\rightarrow 2^A$
- Points in FCAT2 lattice $\rightarrow A^2$

Thus the C2 and FCAT2 structures we use have much lower cardinalities then their logical spaces. For a couple of examples of combinatorics that can be solved, with $D \leq A$:

- Proposition. The number of atomic sequences of length *A,* where $1 \leq A$, that can be chosen from the atomic set *D,* where $1 \leq D \leq A$ is: $Z(A,D) = D(D-1)^{(A-1)} \rightarrow D^A$.
- Proof. The first point in the sequence *A* is chosen from *D* distinct points. Subsequent points are chosen from (D-1) points, as there are no repetitions of the preceding point. Hence the number of distinct sequences is: $D(D-1)\ldots(D-1) = D(D-1)^{(A-1)}$
- Proposition. The number of atomic sequences of length *A* with exactly *D* distinct points, where $1 \leq D \leq A$ is: $Z^*(A,D) = Z(A,D) - Z(A,D-1) = D(D-1)^{(A-1)} - (D-1)(D-2)^{(A-1)}$
- Note $Z^*(A,D)/Z(A,D) = 1 - Z(A,D-1)/Z(A,D) = 1-(D-2)^{(A-1)}/D(D-1)^{(A-2)} \rightarrow 1-1/eD(D-1)$

There are interesting combinatoric problems, but our main concern here will be just with a few questions related to database scalability.

Scalability and Interior Index.

We now pause to illustrate the CAT2 relation in the form of a database table. The remarkable feature (when extended to CAT4) is that a single table structure can represent all types of information with one method, which allows a great simplification of information structures. The essential practical question is whether this is scalable. Here we deal with an important question, which is the scalability of searches or queries of the CAT2 structure, in terms of forming sub-classes of specific information

[4] There is a nondenumerable infinity of increasingly faster functions, including functions faster than any functions it is possible to define recursively.

within larger relations. Although this is put in terms of a database, the *interior index* illustrates the essential theoretical CAT2 property required.

The essential C2 relation has just three columns, here called *ID, ID1, ID2.* This is illustrated below for an example of small relation, with three atoms under the Zero point (which is the only CP in this relation). All other variables, such as coordinates and levels, are determined by the C2 relation determined by these first three columns alone. In the table representation we have added some extra columns, *X, Y,* and *L* to record the coordinates and Levels of points. As these are generally stable, they can be calculated once and recorded semi-permanently for convenient look-up purposes. This is an example of a *denormalized table.*

Table 2.1. Example of a small C2 relation in a denormalized table.

Primary Key	*Left Parent*	*Right Parent*	*Collect Point*	*Atoms*	*X-coord*	*Y-coord*	*Level*
ID	ID1	ID2	CP	A	X	Y	L
0	0	0	0	3	0	3	0
1	0	0	0	3	1	2	1
2	0	0	0	3	2	1	1
3	0	0	0	3	3	0	1
4	1	2	0	3	2	3	2
5	1	2	0	3	2	3	2
6	1	2	0	3	2	3	2
7	2	3	0	3	3	2	2
8	2	3	0	3	3	2	2
9	2	3	0	3	3	2	2
10	2	3	0	3	3	2	2
11	4	7	0	3	3	3	3
12	4	8	0	3	3	3	3
13	4	9	0	3	3	3	3
14	4	10	0	3	3	3	3
15	5	7	0	3	3	3	3
16	5	8	0	3	3	3	3
17	5	9	0	3	3	3	3
18	5	10	0	3	3	3	3
19	6	7	0	3	3	3	3
20	6	8	0	3	3	3	3
21	6	9	0	3	3	3	3
22	6	10	0	3	3	3	3

To populate the columns to the right of *ID2,* we can make a simple recursive function, working down points level by level, since the child properties are determined by parents. In a relational database, we can do this for the entire record set. The query time to join the CAT2 table with itself scales to: *t* ~ log(*N),* with *N* the record count, so this operation is scalable. In practice, we would update these calculated fields each time we add a point. Or we can recalculate for different CP exteriors separately, as the records naturally partition into these. So maintain this index is efficient and very scalable. It is necessary to have such a method, because it is not efficient to keep recalculating the fields from scratch every time we wish to perform an operation employing them.

To run queries across the graph we need to generate the interiors and exteriors of points and determine their intersections. This key operation would be unscaleable if we did it with primitive SQL table joins every time we ran a query. However the C2 interior coordinates along with the *Diamond property* allows an efficient and scalable method. We can make an *interior index table,* as below.

Table 2.2. Interior coordinate index.

Key	*L = 1*		*L = 2*			*L = 3*				*L = 4*				
ID	[0,1]	[1,1]	[0,2]	[1,1]	[2,0]	[0,3]	[1,2]	[2,1]	[3,0]	[0,4]	[1,3]	[2,2]	[3,1]	[4,0]
0	0	0												
1	0	0												
2	0	0												
3	0	0												
4	1	2	0	0	0									
5	1	2	0	0	0									
6	1	2	0	0	0									
7	2	3	0	0	0									
8	2	3	0	0	0									
9	2	3	0	0	0									
10	2	3	0	0	0									
11	4	7	1	2	3	0	0	0	0					
12	4	8	1	2	3	0	0	0	0					
13	4	9	1	2	3	0	0	0	0					
14	4	10	1	2	3	0	0	0	0					
15	5	7	1	2	3	0	0	0	0					
16	5	8	1	2	3	0	0	0	0					
17	5	9	1	2	3	0	0	0	0					
18	5	10	1	2	3	0	0	0	0					
19	6	7	1	2	3	0	0	0	0					
20	6	8	1	2	3	0	0	0	0					
21	6	9	1	2	3	0	0	0	0					
22	6	10	1	2	3	0	0	0	0					

This has the point *IDs* as primary key column, so it joins 1-1 with the primary table of points, above. (This means we can just extend the columns of the primary table, although there may be practical reasons to split tables). Hence it is scalable w.r.t record volume similar to the primary table, having the same record count, and depending on how many columns we want to include, of similar physical size.

The columns represent interior coordinates of the points, and *[0,1]* and *[1,0]* are the immediate parents (*ID1* and *ID2*). Here we show interior points up to 4 levels. As the graph in this case only goes three levels deep, the fourth level coordinates are null. There is of course an unlimited number of coordinates possible, depending on how many levels we want to introduce. But we can only extend the index across in practice to represent a certain finite number of levels. The number of columns goes as: $L(L-1)/2 \rightarrow L^2/2$. E.g. to include 10 levels takes 45 columns. However that is small compared to a free graph, e.g. a binary tree, where the interior positions increase as: 2^L, requiring 1,024 columns for 10 levels. E.g. in CAT2, we could extend the index to 15 Levels, with 15*14/2 = 105 columns, which is still very modest – compared with $2^{15} \approx 32{,}000$ columns for a binary graph.

Let us suppose for practical purposes we adopt a fixed-width index table representing 10 Levels. In fact for the main proposed information representation, we should rarely need more than 10 Levels. However if we do require interiors greater than 10 Levels for some reason, we can extend to 20 Levels by joining the index to itself, which can be done quite efficiently, so the index method is quite scalable in practise.

Also, once we get to the CP level, (0 above) there are no more parents, and values are null. We include the 0 (CP) points in this table, but the final level (the 0's) can be left null, as they are determined by the Level of the point and its CP. Generally, large numbers of coordinate records may be null, and again we can efficiently populate this index and keep it up to date as we add records.

This then gives us an efficient method to generate interior sets, compared to making multiple recursive SQL joins on the base (*ID, ID1, ID2*) table to determine interiors. In fact it reduces processing time for functions like: *p^q* or $p^\vee q$ and *INT(P), EXT(p), INTEXT(p)* to similar scale to: $t \sim log(N)$. Without such a method, repeated SQL joins become exponentially time consuming.

This method is especially effective because of the *Diamond property*. The key operation is to find the exterior intersection of two points *p, q,* under the same CP, as this is the key record set shared by both points. To do this, we first look up the *x,y* coordinates for the points in the previous table ($t \sim log(N)$), obtain the position: *[p]^[q] = [x,y]* and calculate the vector: *[p]^[q] – [p]* ($t \sim constant$). This determines the column in the index row for *p* that has the value *p^q,* if it exists. We do the same for *q,* and compare these values. If they have the same value, this is *p^q.* If not, there is no intersection of exteriors. Hence determining *p^q* is fast and scalable. This extends to multiple points: *p^q^r …*

Without the FCAT2 structure and the *Diamond* property, e.g. for a general graph allowing joins between any two points, this kind of operation becomes very unscalable using simple SQL joins. Graph

DBs use special techniques to try to make this operation scalable, essentially the methods of indexing *chains* connecting points. Our method is faster and simpler.

We then similarly calculate the position *[p]$^\vee$[q]*, and the vectors from this to *[p]* and to *[q]* (*t ~ c*). We then find records in the index where this coordinate column equals *p,* and similarly for *q,* which are two more simple queries (*t ~ log(N)*). We then find common records in these two sets (inner join on two small subsets of *N*, *t~c*). Thus we need only about 5 simple SQL joins, two to find the rows *p* and *q,* two to determine the exterior records for *p* and *q*, and another to match these two sets. This gives us the class: *p$^\vee$q.*

For the full exterior, we then query for the values *p$^\vee$q* in the interior coordinates of points having the same CP, from *p$^\vee$q* to the maximal level: *L[x,y] =* A. This may take the union of several queries, but by filtering on the CP first, this is also very scalable (*t ~ log(N)*). So even without employing other techniques of partitioning sets, this process is very efficient, with overall: (*t ~ log(N)*).

E.g. if it takes 0.001 seconds for an operation with: *N = 1,000,* it should take only about 0.003 secs for *N = 1,000,000,* or 0.006 seconds for *N = 1,000,000,000.* If time scaled linearly with the record set instead: *t ~ N*, it would take about *1 sec* for *N=1,000,000* and about *100* secs for *N=1,000,000,000.* This would be very unscalable.

This addresses a major problem of scalability with graph database representations. The fundamental point is that we need to represent *chains of points* somehow, in a way that is efficient to search for chains from one point to another. While chains are *implicitly represented* in the C2 table (or in graph edge tables), it is not efficient or scalable to make primitive SQL joins to find them – query times will increase unscalably with the table size. As a general principle, we want points represented in contiguous records, or *rows*. This means points are separated in the primitive (binary) data strings by determinate distances. This is how relational tables work: we can find a specific *row* quite efficiently, identified by the row ID, and then we can find the value in a particular *column* of that row efficiently, by moving a determinate number of characters across the row. This is what the asymmetric *row-column* structure of relational database tables provides.

But in general graphs and other network structures, where the class of chains is quite unstructured, there is a vast number of possible chains, making it difficult to represent a table of *chains* that provides contiguous records. Current graph databases require special methods beyond relational database for this. But in CAT2, there are only a limited number of chains upwards from a point into the interior – the number of interior chains scales by: *L(L-1)/2* instead of 2^L**.** So we can effectively represent them in the index table, which has only one row per point. There are still many chains downwards from a point into its *exterior,* but because of the structure typified by the *Diamond property*, it is easy to determine exterior chains within the resources of a relational DB. And the method is quite universal, and easy to program with fully general functions. We continue this in Part 3.

Chains of CPs and Addresses.

Note the full interior chains go above the interior of the CP lattices. But since interior points collect under a single CP, the interior from the CP up is the same for all exterior points. Chains of CPs form trees, which are much simpler to deal with than multiple interior chains. We could make a second index table for this, with only *interior chains of CPs included,* which is a smaller set of points. Again this is represented to only a finite number of levels. But this is simpler than the interior index, and allows a complete calculation of interior and exterior sets, or complete chains.

In this connection we also distinguish these chains from the concept of an *address for a point.* An *address* provides a way to navigate from the top (zero point) to determine a point uniquely. Note that our coordinates do not provide a way of fully specifying what we may call *addresses* for points in the graph. We use integers as names to specify point identities, and in a database table these are primary and secondary keys (IDs). This is a fundamental form of reference to points, but does not represent *addresses*, because point IDs do not represent any relationship between two points.

The problem is to have a method of determining points or classes relative to selected points, e.g. to efficiently determine points at a level or position below another point. Because the exterior is branching, the coordinate positions we have seen determine *classes of points.* We could add an index to specify specific points in the position-classes. E.g. there may be *M* points at position *[x,y]* relative to a CP *z*. If we indexed this class with an integer *m*, we might make 'coordinate addresses' like: *[x,y;m].* This would pick out the m^{th} point in the position *[x,y].* However this does not appear as a feasible method. The graph is dynamic, and whenever it expands or contracts, such indexes would change, and it would be a lot of work to maintain them. Most important, such point addresses do not represent *relationships*, e.g. a point *i* at *[2,1;3]* (the third point at position *[2,1]*) and a point *j* at *[3,1;6]* may have a parent-right-child relationship (they are in the right relative positions for this), but there is no way to tell from this index. Instead, we represent *chains* through the topology.

References.

Note references here include some references for later Parts.

Bridges, Jane. 1977. *Model Theory.* Clarendon Press.

Carnap, Rudolf, 1947, Meaning and Necessity. University of Chicago Press.

Chang, C.C. and H.J. Keisler. 1973. *Model Theory*. North-Holland.

Church, Alonzo. 1956. *An Introduction to Mathematical Logic I.* Princeton.

Codd, Edgar Frank (June 1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM. 13 (6): 377–387. doi:10.1145/362384.362685. S2CID 207549016.

Date, Chris. 2004. (8th Ed.) *An Introduction to Database Systems*. ISBN 0-321-19784-4

Durbin, John R., 1992. *Modern Algebra: An Introduction.* Wiley and Sons.

Duží, Marie. "Intensional Logic and the Irreducible Contrast between De dicto and De re", http://www.cs.vsb.cz/Duží/

Duží, M., Jespersen, B., Materna, P. 2010. *Procedural Semantics for Hyperintensional Logic: Foundations and Applications of Transparent Intensional Logic*. Springer Verlag.

Frege, Gottlob, 1879. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle a. S.: Louis Nebert. Translation: Concept Script, a formal language of pure thought modelled upon that of arithmetic, by S. Bauer-Mengelberg in Jean Van Heijenoort, ed., 1967. From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931. Harvard University Press.

Frege, Gottlob, 1884. Die Grundlagen der Arithmetik: Eine logisch-mathematische Untersuchung über den Begriff der Zahl. Breslau: W. Koebner. Translation: J. L. Austin, 1974. The Foundations of Arithmetic: A Logico-Mathematical Enquiry into the Concept of Number, 2nd ed. Blackwell

Frege, Gottlob, 1892. "On Concept and Object." (First published in the Vierteljahrsschrift fir wissenschaftliche Philosophie, 16 (1892).

Holster, Andrew T. 2008-2011. "System and method for representing, organizing, storing and retrieving information." US. Patent Number: 7,979,449. July 12, 2011.

Materna, P. 2004. Conceptual Systems. Berlin: Logos.

Materna, Pavel. 1998. *Concepts and Objects.* Acta Philosophica Fenica, vol. 63, 1998.

Montague, R. 1970. "Universal Grammar". *Theoria* **36,** pp. 373-398.

Montague, Richard. 1973. "The Proper Treatment of Quantification in Ordinary English". *Approaches* *to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics.* D.Reidel.

Robinson, Ian, Webber, Jim and Eifrem, Emil. 2015. *Graph Databases (2nd Edition)*. NEO4J.

Russell, Bertrand. 1905. "On Denoting", Mind, Vol. 14. ISSN 0026-4423. Basil Blackwell.

Tichý, Pavel. 1971. "An Approach to Intensional analysis", *Nous* **5,** pp. 273-297.

Tichý, P. 1988. The Foundations of Frege's Logic. Walter de Gruyter.

Tichý, P. 2004. Pavel Tichý's Collected Papers in Logic and Philosophy. Svoboda, V., Jespersen, B., Cheyne, C. (eds.), Dunedin: University of Otago Publisher, Prague: Filosofia.

TIL (Transparent Intensional Logic) Website: http://www.phil.muni.cz/fil/logika/til/index.html

van Benthem, Johan and Alice ter Meulen. 1997. *Handbook of Logic and Language.* M.I.T. Press.

Wittgenstein, Ludwig. Philosophical Investigations , 1953, G.E.M. Anscombe and R. Rhees (eds.), G.E.M. Anscombe (trans.), Oxford: Blackwell.

Wittgenstein, Ludwig. Tractatus Logico-Philosophicus (TLP), 1922, C. K. Ogden (trans.), London: Routledge & Kegan Paul. Originally published as "Logisch-Philosophische Abhandlung", in Annalen der Naturphilosophische, XIV (3/4), 1921.

Wikipedia pages.