

WHY NOT SETS?

Abstract

Sets are well defined collections of uniquely identifiable items. Data used by computers are well defined collections of items representing situations of interest. Computers themselves are just well defined collections of bits that change value over time. It would seem that all computer processing is highly set oriented. Why are sets not more widely used in modeling the behavior and assisting the development of computing systems? The following dialogue will attempt to amplify this question, though neither of the participants has a clue to the answer.

I. MODELING SYSTEMS BEHAVIOR WITH SET THEORY?

Quizzer: Could sets have a larger role in modeling the behavior and assisting the development of computer systems than they currently do?

Authority: Explain what you mean by 'larger'?

Q: Why not use sets to model data representations and set operations to model data processing?

A: Could you be more specific?

Q: Can't all computer execution states be represented by a string of bits, and can't the behavior of a computer be modeled by the succession of one state of bits to another state of bits?

A: Supposing this is true, what is your point?

Q: Can't all strings of bits be represented by a sequence, or tuple, and can't the behavior of a computer be modeled by the mapping of one bit-tuple to another bit-tuple over time?

A: No offense, but this is not a startling revelation.

Q: Sorry, I am not a computer scientist. What I was trying to ask was, if tuples are sets and if all computer behavior can be modeled by a mapping of sets to sets over time, then why isn't all computer behavior a form of set processing?

A: Who ever said it wasn't?

Q: Let me rephrase the question. If all computers are intrinsically set-processing systems and if all software can be modeled with mathematical precession using sets and operations on sets, then why is set-theoretic modeling not the *de facto* standard of the computing industry?

A: Just what would you expect would be achieved if this were so?

Q: Like any other engineering discipline that uses mathematical foundations to guide construction, I would expect more reliable systems behavior, more capability, and better performance control.

A: We have to be careful here. *Reliable behavior* implies that the hardware component of a system executes precisely and consistently what the software component dictates and since this is always the case, systems can not be more reliable than they already are.

Q: What I mean by *more reliable* is that the software component more closely dictates desirable behavior than it currently does.

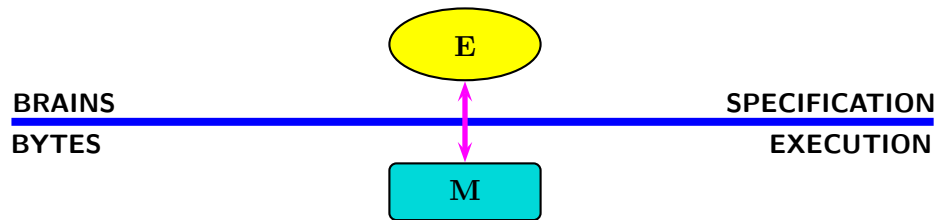
A: Now we have room to maneuver. Two factors come into play: 1) how accurately does the software capture the specification of the desirable behavior; and 2) how well does the software component communicate these specifications to the hardware component?

Q: That seems reasonable, but how does that refinement help?

A: In my opinion, the best way in which a formal model of system behavior can be of value is to breach the *brain/byte* schism between application specification and system execution. This refinement pinpoints exactly that issue.

Q: How does this help explain the connection between specification, execution, and set theory.

A: Computer systems implementations rely on a negotiated agreement between application specification and system execution. Often the focus is just on what has to be done and not on how *best* to do it. We need to model both. Perhaps a visual aid might be helpful. The diagram below depicts the implementation schism between the brain world of people and the byte world of computers.



This simplistic diagram represents the basic problem of modeling computer systems behavior. The Enterprise (**E**) represents the world of people and applications. The Machine (**M**) represents the world of processors and storage. These two worlds are *disjoint* in every sense of the word.

Q: I like your colorful little picture, but I'm not sure I appreciate your use of the term *disjoint*.

A: It means they do not intersect and that what is *friendly* to one is *antagonistic* to the other.

Q: That makes sharing anything friendly between them, like data, rather difficult.

A: A proven fact, but I propose that there is one mutually friendly exception.

Q: Yes?

A: Set membership.

Q: Set membership? How do you explain that?

A: With light switches.

Q: With light switches?

A: The simplest possible computer is the light switch. It has two logical states: 'on' and 'off'. No need for a formal model, the hardware will fail before the logic does. The next complicated computer is a pair of light switches, giving four logical states. Again the hardware will fail before the logic does, as long as all four states are accountable. Given a sequence of n binary switches there will be 2^n logical states to account for. As long as all 2^n are accounted for, the hardware will fail before an unpredictable state occurs. As n gets large, the likelihood of logical error occurring before the hardware fails becomes more likely. The issue here is one of complexity accountability, not one of computational difficulty.

Q: Are you claiming that if we had a formal model that could account for all possible states, we would never have a software failure? That would certainly qualify as a reliable system.

A: Almost. If we had a formal model that would only allow acceptable states to be generated and if these conditions were preserved by the programs generated, we would never have a software failure.

Q: So you're not addressing problems of NP-completeness, decidability, program correctness, nor any other application specification difficulties. You're only addressing the issue of ensuring that the application specified is actually the one being executed by the computer.

A: Exactly, and using set theory for validation. The point here is that every software specification expressible in a computer and every machine module of executable code has a bit-string, or tuple, representation. Thus, since tuples are sets and the transformation of sets to sets is a set-theoretic process, the modeling of software specifications and their embedding into an executable form by means of set theory is a reasonable expectation.

Q: So why is set-theoretic systems modeling not an actual reality?

A: Set theory, *as it is generally known*¹, is not up to the task.

Q: That makes no sense. Set theory is adequate enough to model all of known mathematics.

A: The key word here is *MODEL*. What mathematics can model in terms of abstract truths and what is required to be modeled for software development in terms of computing functionality and performance are very, very different tasks.

Q: Could you elaborate on that?

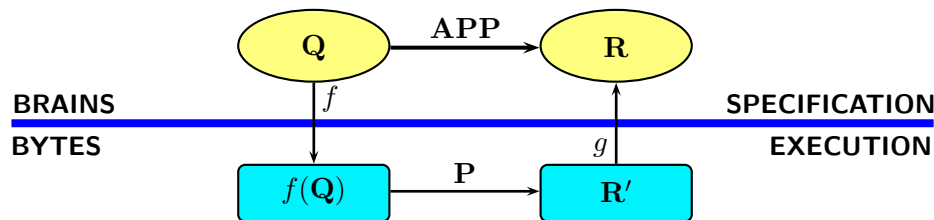
A: Sure. Software systems implementation is a Janus discipline. You have user application specification concerns on one side and system execution performance concerns on the other. Given the luxury of mathematics' ability to ignore time and money there is no perceptible problem. However, finite resources and time critical applications make performance issues necessary considerations for any system implementation model.

Q: That doesn't explain the connection between specification, execution, and set theory.

A: Let's look more closely at what is actually required of a systems model. First the desired result of an application has to be *precisely* stated in the enterprise environment. Second, the application specification has to be *precisely* embedded into the machine environment.

Q: Why did you emphasize *precisely*?

A: That indicates where set theory needs to be involved. Our visual aid can help illustrate this.



In the above diagram \mathbf{Q} and \mathbf{R} are user-friendly data representations with $\mathbf{R} = \mathbf{APP}(\mathbf{Q})$. \mathbf{Q} is embedded by f into a machine-friendly data representation $f(\mathbf{Q})$. The executable equivalent of an application, \mathbf{APP} , is \mathbf{P} , producing machine-friendly representation \mathbf{R}' . The *extraction* $g(\mathbf{R}')$ transforms \mathbf{R}' into the user-friendly representation \mathbf{R} . Thus, $\mathbf{R} = g(\mathbf{P}(f(\mathbf{Q}))) = \mathbf{APP}(\mathbf{Q})$. In general this commutativity expression would deny comprehension by mere mortals. Thus, user friendly languages with strong formal underpinnings need to be built to manage the complexity.

Q: Interesting. With a formal model, system reliability could be assured if for all legitimate values, the diagram commutes. Why does this reflect badly on set theory?

A: In the diagram there are four required data sets and four required set operations. Set theory can not *faithfully represent* all of these data sets and therefore not all of the required operations.

Q: I notice you stressed *faithfully represent*. Why?

A: It is an essential concept in the sense that these sets do not have a mathematical identity under

¹Those set theories using nested sets to define tuples like $\langle a, b \rangle = \{\{a\}, \{a, b\}\}$ due to Kuratowski.

set theory and therefore cannot be considered to be mathematical objects and thus can not be considered as operands to any set operation.

Q: You threw a lot of terminology at me just now. What do you mean by *mathematical identity*?

A: In our context *mathematical identity* means that an item is recognizable as a well-defined set that preserves all properties of interest.

Q: I still don't see why this is troublesome for set theory. It is a common programming practice to use *records* for both logical and physical data representations, or what you refer to as specification and execution environment data representations. Since records have a well defined representation as *tuples* and since all transformations between logical and physical representations are transformations between collections of records (tuples), I don't see any problems with using set theory.

A: Ah, the ubiquitous use of tuples to model records. You could not have chosen a better nemesis to set theory. *TUPLES ARE NOT WELL-DEFINED MATHEMATICAL OBJECTS!*²

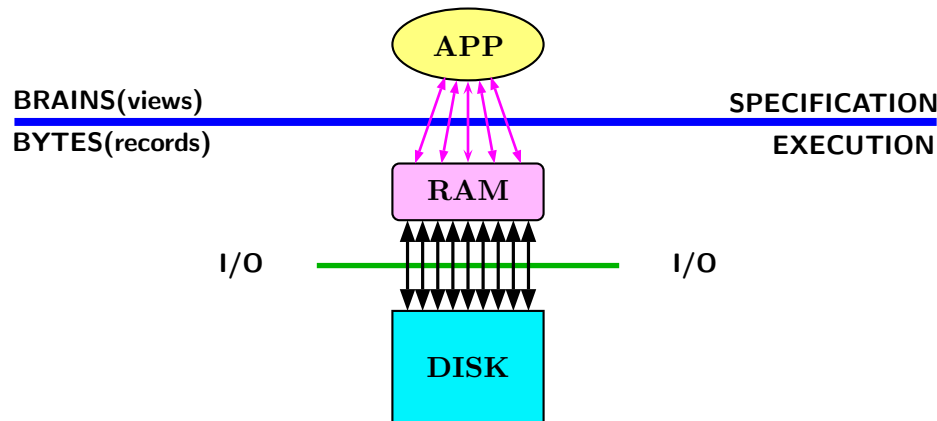
Q: You seem pleased.

A: Only in that you have so quickly identified the primary reason that set theory has been rejected as an implementation modeling paradigm.

Q: How so?

II. MODELING REQUIREMENTS OF SET THEORY

A: For set theory to be of any value for modeling the behavior and assisting the development of computing systems there is only one real requirement, being able to treat any and all data representations as mathematical objects. Let's expand our visual aid to more accurately reflect the types and interrelationships of the data representations involved.



The above diagram reflects a more realistic operational system with three distinct classes of data representation: presentation, processing, and preservation.

Presentation data (commonly referred to as user views) is the abstract, informal representation of data suitable for manipulation by programmers and languages for application specification and usage.

Processing data is a machine friendly (byte-sequence) representation of presentation data in a form suitable for manipulation by a CPU.

Preservation data is a machine friendly (byte-sequence) representation of processing data in a form suitable for long term storage and rapid I/O access.

Data transformations between these representations can be extremely complex.

Q: So both abstract user views and concrete machine representations have to be formally modeled.

²Justification for this statement can be found in [Ch95], also see Skolem [Sk57].

A: Correct, but most importantly the operations, that user-friendly languages bounce off the abstract informal user views and that the underlying support mechanisms are trying to execute, have to also be formally modeled.

Q: I can see where this might be a considerable challenge. Is there a simple example that goes from user to processor to storage and back?

A: Sure. Let's start with the simplest of all possible user views of data, the array:

$$\mathbf{H} = \begin{array}{|c|c|c|c|} \hline h & v & w & t \\ \hline d & b & a & c \\ \hline w & x & y & z \\ \hline \end{array}$$

and to make life simple, for now, assume that the vales of the array are one byte characters.

Q: If I'm following you correctly, your next step is to provide \mathbf{H} with a mathematical identity, then show the RAM resident preservation of this identity.

A: Absolutely correct. Any suggestions for a candidate identity for \mathbf{H} ?

Q: You'd be disappointed if I didn't suggest a 3-tuple of 4-tuples.

A: Right again and let's assume for the moment that we have a definition for an n-tuple that behaves the way we want it to. So, we will assert by the notation ' $\mathbf{S} = \langle x_1, \dots, x_n \rangle$ ' that \mathbf{S} is a sequence of items x_1 through x_n , or equivalently, that x_i is an 'i-th' element of \mathbf{S} . Using this hypothetically formal notation for n-tuples, we can *formally* define \mathbf{H} by:

$$\mathbf{H} = \langle \langle h, v, w, t \rangle, \langle d, b, a, c \rangle, \langle w, x, y, z \rangle \rangle .$$

Which, as you predicted, is a 3-tuple of 4-tuples.

Q: And now the RAM representation?

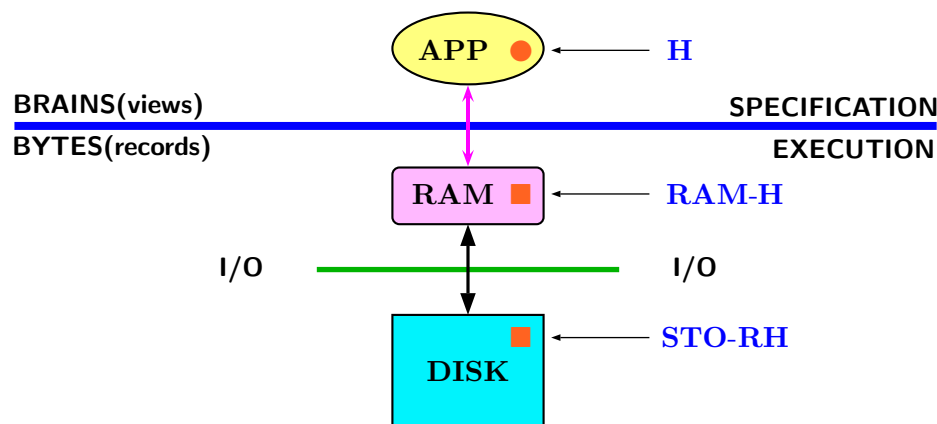
A: And now the RAM representation, this time using $\langle b_1, \dots, b_k \rangle$ as a sequence of bytes, giving:

$$\mathbf{RAM-H} = \langle h, v, w, t, d, b, a, c, w, x, y, z \rangle$$

An easy mapping since \mathbf{H} entries were chosen as characters. More complicated entries would result in a longer byte-sequence for $\mathbf{RAM-H}$.

Q: I assume, in this simple case, that the DISK representation would be the same as in RAM.

A: Why not? By setting $\mathbf{STO-RH} = \mathbf{RAM-H}$ our visual aid might look like:

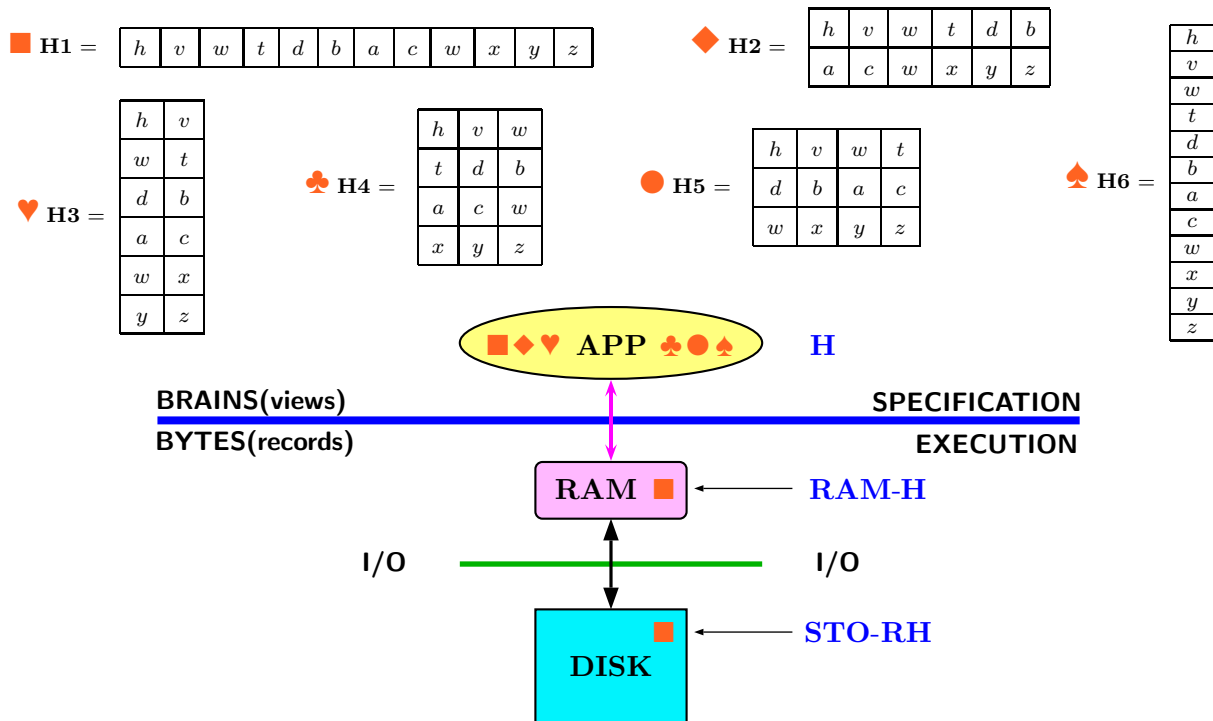


Q: I notice that you used a little orange dot for \mathbf{H} instead of a square. Was that intentional?

A: Good eye, indeed it was. Notice \mathbf{H} is not equal to $\mathbf{RAM-H}$, but is one of many interpretations of $\mathbf{RAM-H}$ as an array. Information contained in \mathbf{APP} transforms $\mathbf{RAM-H}$ into \mathbf{H} .

Q: One of many?

A: Actually, one of 6 different arrays. In fact, it might be worth looking at all 6.



Notice that all 6 user views have the same representation in RAM. Knowledge in the program that uses the RAM representation must pick which view is appropriate.

Q: Let me ask a few questions to make sure I appreciate this formal delineation between brains and bytes which we are in the habit of munging together. All the **Hs** are abstract, informal, non-byte oriented, user-friendly conceptual aids for guiding our manipulation of machine resident operations that mimic our imaginary manipulations of the user views.

A: Absolutely correct and unfortunately antithetical to common practice. For example, in practice **H1** and **H6** would be treated differently than the other **Hs**.

Q: How so?

A: **H1** would likely be treated as a 12 place record, while **H6** would be treated as 12 entry list.

Q: What's wrong with that?

A: Nothing, it just ignores the possibility of keeping them in the same family of arrays with formal representations like:

$$\mathbf{H1} = \langle\langle h, v, w, t, d, b, a, c, w, x, y, z \rangle\rangle, \text{ and}$$

$$\mathbf{H6} = \langle\langle h \rangle, \langle v \rangle, \langle w \rangle, \langle t \rangle, \langle d \rangle, \langle b \rangle, \langle a \rangle, \langle c \rangle, \langle w \rangle, \langle x \rangle, \langle y \rangle, \langle z \rangle\rangle .$$

Q: Why would one want when it is so much easier to view **H1** and **H6** the way they are represented in RAM?

A: Its only easier with trivial examples. The approach does not scale up with complexity. What is simple for three light-switches is not so simple for three million. Examples, by their very nature, have to be simple. This was a most simple example of data representation mappings between brain and byte environments. Yet even it was not without complexities. Imagine what happens when programmers and language designers have to deal with really interesting user views?

Q: I see your point.

A: There is another, less obvious but very important point. As we discovered with this example, there were really 6 user views supported by the same RAM. Each of these had distinct mathematical identities. Which means there were 6 distinct mathematical objects. Different mathematical objects can behave differently under the same mathematical operation. Array operations, list operations, and record operations need to match with their operands. Mixing the **Hs** could make language and programming development more complicated than it need be.

Q: Could you give a more complicated example showing where formal control of the mapping between brains and bytes makes life easier for program and language developers?

III. RELATIONAL MODEL

A: Sure, are you familiar with the Relational Model of Data (RDM)[Co70]?

Q: Very.

A: You have no problem with terms like: Data Independence, normalization, RDM-relation, RDM-table, RDM-relationship, RDM-domain and RDM-operations?

Q: None whatsoever.

A: Very good. Let's follow the simple array example and start with a simple RDM-table, its RAM representation, and then the storage representation of the RAM representation.

Q: That would be good.

A: Allow me a pedagogical test, what can you tell me about the following:

$$\mathbf{T} \rightsquigarrow \begin{array}{|c|c|c|} \hline & A & B & C \\ \hline a & b & c \\ \hline x & y & z \\ \hline \end{array}$$

where ' \rightsquigarrow ' is to be read as 'preserves all properties of interest'.

Q: Assuming that its a trivial example of an RDM-table, I would tell you that it is a user-friendly view of a RDM-relation, where the RDM-domains are represented by A , B , and C . The represented RDM-relation is a ternary relation containing two 3-tuples. Did I pass?

A: Sorry, I was not trying to be demeaning. So often pseudo-experts assert vast knowledge of subjects they have no real deep understanding about, that I get overly sensitive. Yes, you passed.

Q: No offense, I'm familiar with that breed. Do continue.

A: The RDM-table, \mathbf{T} , is the user view used as a virtual operand for RDM-operations. For any operations to actually execute, there must be a RAM representation that captures the mathematical identity of \mathbf{T} . There is no criteria for uniqueness, but the representation has to be in terms of byte sequences with an implicit understanding (part of a program) on how the mathematical identity of \mathbf{T} is related to the RAM representation. Following is such a candidate:

$$\mathbf{RAM-T} = [A, B, C, a, b, c, x, y, z]$$

where $\mathbf{RAM-T}$ represents a 9-byte sequence somewhere in RAM.

Q: This is what I would expect, given your previous array example.

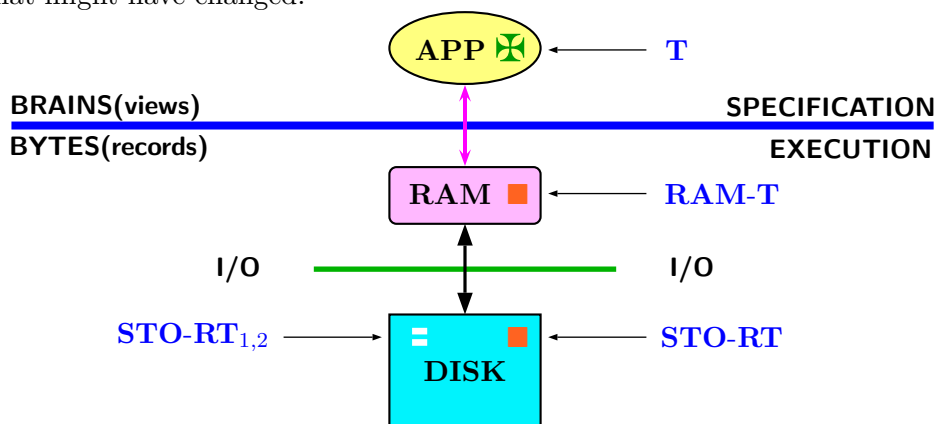
A: As in the array example we can map $\mathbf{RAM-T}$ directly to $\mathbf{STO-RT}$, and in order to make this example a little more interesting I'm also going to split $\mathbf{RAM-T}$ into two storage representations requiring an operation to reconstruct $\mathbf{RAM-T}$.

$$\mathbf{STO-RT}_1 = [A, B, C, a, b, c] \quad \text{and} \quad \mathbf{STO-RT}_2 = [A, B, C, x, y, z]$$

The program using these would know that they would have to be combined to form **RAM-T**.

Q: Diagrammatically this would not be much different than the simple array example, would it?

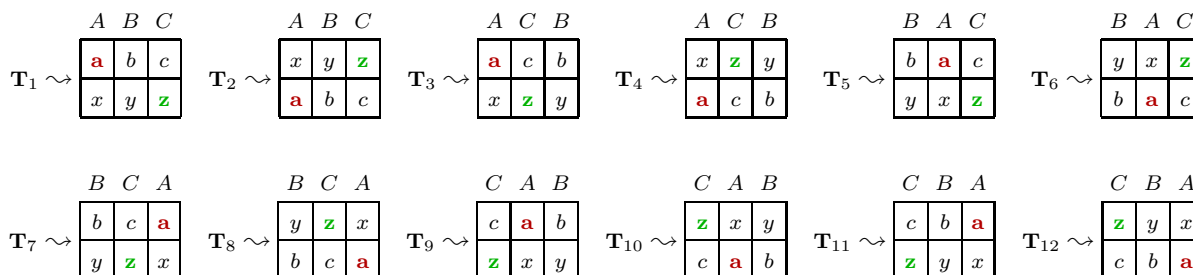
A: Visually not by much, but mathematically there may be some surprises. Let's revisit our visual aid to see what might have changed.



As in the array example, the **RAM-T** to **STO-RT** represents the familiar and unimaginative WYSIWYG data access mapping, while the **RAM-T** to **STO-RT_{1,2}** provides a more interesting relationship between RAM and DISK, which requires operations to decompose and reconstruct the RAM representation (depicted by f and g in the earlier commutativity digram).

Q: I know its intentional, so I assume your green iron cross implies a need for special attention.

A: Indeed it does. **T** represents an RDM-table, not a simple array. Here is where things start to get really interesting. Let me ask you a question, what can you tell me about the following:



Q: Seems like we've done this before. All twelve are distinct RDM-tables, each one representing the exact same RDM-relation.

A: How can you tell?

Q: RDM-tables are labeled arrays used as informal representations of an RDM-relation. Since there are 12 distinct arrays, there are 12 distinct expressions of the same represented RDM-relation. Since the labels are the same, and since both the rows and the columns are respectively in one to one correspondence, the represented RDM-relation is the same for all 12 distinct representations.

A: Perfect. In terms of a formal modeling, you have asserted that there are 13 distinct mathematical objects, 12 of which contain the mathematical identity of the remaining object. True?

Q: True, if it can be shown that the 13 mathematical identities actually exist.

A: Given our hypothetically formal n-tuple notation from the array example, could you construct the 13 necessary mathematical identities?

Q: Possibly, what are the ground rules?

A: As mathematical objects all properties of interest have to be derivable using operations. In this case four operations are required to validate the mathematical identities:

- 1) An operation to extract the domain names from RDM-tables.
- 2) An operation to extract row-values from the RDM-tables.
- 3) An operation to extract column-values from the RDM-tables.
- 4) An operation to extract the RDM-relation from the RDM-tables.

Q: So what does a hypothetically formal set operation on a hypothetically formal n-tuple look like?

A: To answer that we need to equate our hypothetical n-tuple with a hypothetical set. A simple means for doing that is to make special use of the superscript notation, as in the following:

$$\mathbf{S} = \langle x_1, \dots, x_n \rangle = \{ x_1^1, \dots, x_n^n \}$$

asserting that \mathbf{S} is an n-tuple of items x_1 through x_n , and that x_i is an ‘i-th’ element of \mathbf{S} . Using this extended notation for defining sets, we can redefine array \mathbf{H} which was:

$$\mathbf{H} = \langle \langle h, v, w, t \rangle, \langle d, b, a, c \rangle, \langle w, x, y, z \rangle \rangle,$$

by:

$$\mathbf{H} = \{ \{h^1, v^2, w^3, t^4\}^1, \{d^1, b^2, a^3, c^4\}^2, \{w^1, x^2, y^3, z^4\}^3 \}.$$

For now we can refer to this hypothetically formal notation for defining sets as extended set notation (XSN). At this point we should consider XSN just a convenient notation without supporting rigor³.

Q: So the basic idea behind XSN is to augment Classical set notation (CSN) by using integers as element superscripts to indicate an ordering relationship between elements of a set?

A: The motivation for using superscripts is to give set membership an extra degree of qualification. When set membership is only intended to reflect the properties required of an n-tuple, yes. However, we have not, as yet, imposed any restriction on what the scope of element superscripts could be.

Q: What else would be meaningful besides integers?

A: You mean, what other properties would be of interest besides order?

Q: If you prefer.

A: Let’s examine the possibilities. CSN expresses the condition that for any given set, an item is either an element of that set or it is not an element of that set:

$$\mathbf{CSN:} \quad \mathbf{Q} = \{x\} \quad \text{means ‘}x\text{’ is an element of } \mathbf{Q}.$$

XSN expresses the condition that for any given set, an item is either a y-property element of that set or it is not a y-property element of that set:

$$\mathbf{XSN:} \quad \mathbf{Q} = \{x^y\} \quad \text{means ‘}x\text{’ is a }y\text{-property element of } \mathbf{Q}.$$

When the conditional property is the relative ordering of the set elements, we have tuples, but since mathematics is more concerned with the correctness of expression than with the meaningfulness of that being expressed, there is no need to restrict the scope of values that can be assumed by ‘y’ as long as no antinomies are generated.

Q: You have not provided any properties of interest. You have only provided an argument for expressing them with XSN, if we can think of any.

A: Any come to mind?

Q: Could XSN element superscripts be RDM-domain names?

³‘What we lack in rigor we shall make up in notation.’[Ma83] As David Maier wittingly pointed out, sometimes our only option is to develop a notation as best we can, when formal derivation is not available as a guide. Fortunately, XSN has axiomatic support [Ch95].

A: Why not, so long as our use of XSN does not produce a contradiction? Let's refer to the term 'element superscripts' as *scopes*, and using scopes to reflect properties of interest let's see what an RDM-relation might look like as an extended set:

$$\mathbf{T} = \left\{ \{a^A, b^B, c^C\}, \{x^A, y^B, z^C\} \right\},$$

which I think you will recognize as the RDM-relation represented earlier by each of the RDM-tables \mathbf{T}_1 through \mathbf{T}_{12} .

Q: Interesting. I can see where the set-theoretic notation captures all the relevant properties for this simple example, but for any real commercial application a user or programmer would get lost in the set brackets before ever being able to use it.

A: God forbid! We are addressing the issue of the mathematical identity of RDM-relations so that they may be used as mathematical objects at the execution level. No programmer nor user should ever be aware that mathematical rigor crept into the support of their application.

Q: That's a relief. I know from experience that set-theoretic notation can cause sever migraines. So users can still use informal RDM-tables to express the operations they want to do, but under the covers real set-theoretic operations would be applied to \mathbf{T} .

A: Right, but in order for formal operations on RDM-tables to be mapped to formal operations on RDM-relations, all unique RDM-tables must be unique mathematical objects.

Q: Makes sense. So what might the mathematical identity of an RDM-table look like?

A: An RDM-table is an abstract informal user-friendly way of representing an RDM-relation and in that sense all RDM-tables that represent the same RDM-relation are *content equivalent*. However, since RDM-tables are represented as labeled arrays, content equivalent RDM-tables will be *structurally distinct*. Both of these properties have to be captured by the mathematical identities of RDM-tables. Since we are modeling with XSN the mathematical identities must be specified by set membership. From the 12 RDM-tables in our example, represented as labeled arrays, \mathbf{T}_1 and \mathbf{T}_{12} might look like:

$$\begin{array}{c} \mathbf{T}_1 \rightsquigarrow \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ \hline x & y & z \\ \hline \end{array} \quad \mathbf{T}_1 = \left\{ \{a^{<1,A>}, b^{<2,B>}, c^{<3,C>}\}^1, \{x^{<1,A>}, y^{<2,B>}, z^{<3,C>}\}^2 \right\} \\ \\ \mathbf{T}_i = \langle \mathbf{LabeledRow1}, \mathbf{LabeledRow2} \rangle \\ \\ \mathbf{T}_{12} \rightsquigarrow \begin{array}{|c|c|c|} \hline C & B & A \\ \hline z & y & x \\ \hline c & b & a \\ \hline \end{array} \quad \mathbf{T}_{12} = \left\{ \{z^{<1,C>}, y^{<2,B>}, x^{<3,A>}\}^1, \{c^{<1,C>}, b^{<2,B>}, a^{<3,A>}\}^2 \right\} \end{array}$$

Both \mathbf{T}_1 and \mathbf{T}_{12} are content equivalent in that they both represent the same RDM-relation, but \mathbf{T}_1 and \mathbf{T}_{12} are structurally distinct in that their rows and columns are not identical. The mathematical identities of \mathbf{T}_1 and \mathbf{T}_{12} capture both the content equivalence and the structural distinctions in such a way that they can be set-theoretically separated.

Q: It looks like you use scope values to carry meta-data information about the actual data items. Your 'labeledrow' looks like what some authors would call a 'labeled n-tuple', which you have captured with XSN by using a 2-tuple, containing order and domain name, as a scope.

A: Correct. Let's examine these scopes a little more closely. Both \mathbf{T}_1 and \mathbf{T}_{12} have two elements which reflect the two records in the RDM-tables. The scopes of these two elements reflect the row order in the labeled array representation of the RDM-tables.

Q: Interesting. I also notice that just the ordering information differs, but the domains for respective elements remain the same.

A: Notice also that setting all the order values to 0 gives:

$$\mathbf{R} = \left\{ \{a^{<0,A>}, b^{<0,B>}, c^{<0,C>}\}^0, \{x^{<0,A>}, y^{<0,B>}, z^{<0,C>}\}^0 \right\},$$

which easily reduces to :

$$\mathbf{T} = \left\{ \{a^A, b^B, c^C\}, \{x^A, y^B, z^C\} \right\}.$$

Its pretty simple stuff, once you get the hang of it.

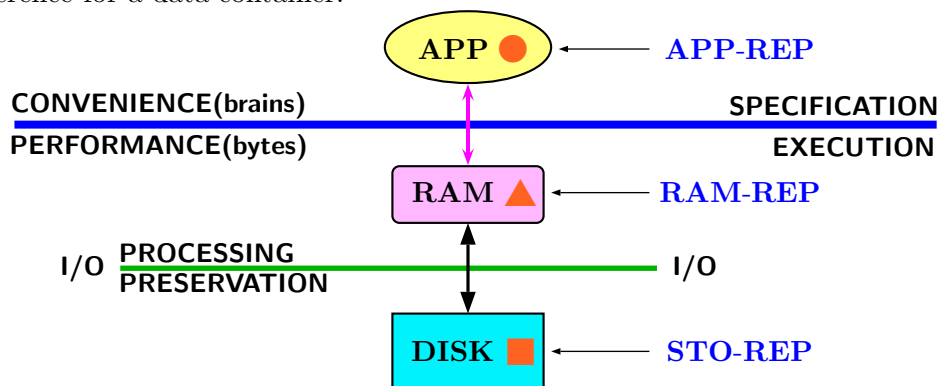
Q: Of that, I am not yet convinced. Though XSN does seem to offer a degree of abstract representation precision not available with CSN, it is not obvious to me now this abstract precision provides any practical value in the implementation and use of systems.

IV. DATA CONTENT VS. DATA CONTAINER

A: By expressing data representations as mathematical objects, as we just did with the RDM-tables and RDM-relations, the *data content* became distinguishable from the *data container*. Notice, at the specification level using RDM-tables and RDM-relations, the data content (the RDM-relation) of all twelve different data containers (the RDM-tables) was the same.

Q: That is quite clear, but does not address my concern about implementation value.

A: Actually, it does, once you fully appreciate the ability to separate data content from data containers. Recall our visual aid has three distinct environments for representing data. Each has its own preference for a data container.



Notice here that the data content (orange) is the same in all three environments, but the data containers (●, ▲, ■) are different. The implementor's problem is to capture relationships between data items in a brain friendly representation at the specification level and ship this representation off to the execution level in such a way that it arrives in a byte friendly form.

Q: I can understand the need for different data representations for specification convenience and for execution performance because of the brain/byte interface, but I don't understand the need for different data representations between RAM and DISK, since both are byte environments.

A: Good observation, in fact traditional systems don't usually make any distinction.

Q: So why are you making the distinction?

A: For improved performance control.

Q: I don't follow.

A: For traditional transaction processing, where the I/O requirement is to access a single atomic record in storage as quickly as possible and plop it into RAM for processing, an identical record for both RAM and DISK is the ideal data container. However, these traditional record I/O access architectures are grossly inadequate for the modern requirements of extracting information from many multiples of records from distributed disparate data sources.

Q: So records are inadequate as shipping containers for modern day I/O access requirements?

A: Quite the contrary, records are ideal for both traditional record I/O access requirements and for today's more demanding I/O access requirements. Let me explain. Traditionally, records have only been considered as physical objects, not as mathematical objects. As physical objects, records are accessed and manipulated by the location of the data container. As mathematical objects, records are accessed and manipulated by the content of the data container.

Q: Okay, I buy that physical objects are atomic and can only be moved around as complete units. I also appreciate that mathematical objects can be decomposed, restructured, combined, and generally manipulated by appropriate operations. What I don't yet appreciate is what such a mathematical object might look like and what operations on it would make any practical difference.

V. I/O PERFORMANCE

A: Fair enough. To make a substantive case for how mathematical objects and operations on them that can improve control over I/O performance we first need to establish some metric for measuring I/O performance.

Q: I/O performance issues are quite complex and have been the study of intense investigation for over forty years. How can you establish any comprehensive conversational metric?

A: By ignoring all implementation details and focusing on the basic performance essentials.

Q: The Carnot heat engine approach?

A: Exactly, establish an unattainable optimum and compare candidate systems to the optimal. For our purposes a metric is quite simple. We can define optimal I/O performance for a specific application to be the total time it takes to transfer all relevant data at the highest data transfer rate (DTR) available to the application.

Q: Simple enough, though it ignores many hidden costs, like data loading, index construction, number of disk accesses, and more.

A: That's okay. We're just using it to analyze what I/O access strategies are available if data representations were treated as mathematical objects instead of as physical objects.

Q: So what considerations are critical to I/O performance?

A: We have two application types to consider: transaction processing applications and information extraction applications. It is well established that for transaction processing atomic records and indexed record I/O access is as close to optimal as one can get. However, for information extraction applications the case is quite different. Many, many records may have to be accessed during the execution of a specific application, but only a very small part of the records accessed may be relevant data. Typically only of 5% to 10% of the content of a record is relevant data for a specific application and it is seldom the same content for the next application.

Given that indexed record I/O access techniques use I/O transfer buffers that are only 80% full of data and that they typically use random access DTRs which are at best a quarter as fast as sequential DTRs. Then 10% of 80% is 8% and a quarter of that is 2% giving indexed record I/O access techniques a 2% of optimal performance for information extraction applications.

Q: I never realized indexed record I/O access was so slow.

A: Hold on. Nobody said indexed record I/O access was slow. First of all it is the fastest known for transaction processing applications and secondly, 2% of an ideal unattainable optimal may be the best that can realistically be achieved. Especially if the only other option is doing an exhaustive search of all relevant and non-relevant data

Q: You're obviously implying that exhaustive searching and indexed record I/O access are not the only two options available.

A: Let's explore what the characteristics of a potential third option would need to be in order to improve I/O performance.

Q: Just using sequential DTRs would bring the 2% up to 8%.

A: True, and that 8% represents the percent of relevant data, or the *information density*, of an I/O transfer. So the objective of a third I/O access alternative would be to raise, as high as possible, the information density of I/O transfers. This, of course, requires dynamic restructuring of data representations during the execution of an application and that requires the I/O access process to be a mathematical transformation of data between RAM and DISK.

Q: I thought that might be where you were headed, but mathematical transformations also require mathematical objects to be transformed. What might such an object look like at the I/O interface?

A: The best way to explain what RAM/DISK friendly data representations could look like might be to look at Codd's user-friendly view of representing data, but upside down. Codd introduced *labeled arrays*, or tables, to provide a visual representation of data relationships for the convenience of users. These labeled arrays were physical objects without a mathematical identity. We need to define an abstract RAM/DISK friendly mathematical object reflecting physically represented data in a machine environment.

Q: I'm not sure I completely follow what you just said.

A: The idea here is to do on the machine side what Codd did on the user side. That is to provide an abstract machine-friendly representation of physical data as a guide for choosing appropriate operations to manipulate that data, just as the abstract RDM-tables provide a guide on the user side.

Q: What exactly are the requirements for an abstract representation of physical data?

A: For operations to be performed at the I/O interface compatible operands must exist on both sides of the interface. These operands need mathematical identities that faithfully reflect the data representations of the RAM and DISK environments.

Q: So you need Codd's form of data independence⁴ at the I/O interface.

A: Exactly, and where would be a better a better place to start than to examine an I/O interface that supports RDM-relations and the RDM operations on them?

Q: RDM operations on physical data at the I/O interface?

A: That's the idea.

Q: Does that mean that RAM data representations would be data independent of DISK data representations?

A: Yes, but this is not as impossible as it may seem, if we are given an adequate description of the mathematical identity of objects being exchanged across the I/O interface.

⁴As defined by Codd, absolutely no knowledge of the underlying physical representation of the data.

Q: You have a candidate of course?

A: More than one, actually. Let's start with a simple but adequate set-membership expression of RAM/DISK data representations. We can build on Codd's user-friendly labeled array idea, but instead of it being an actual array with rows and columns, it will be an array-resembling set-membership description. Following is an example of a *doubly labeled array* (DLA).

$$\begin{array}{c} A \ B \ C \\ \mathbf{r} \begin{array}{|c|c|c|} \hline a & b & c \\ \hline \end{array} \\ \mathbf{s} \begin{array}{|c|c|c|} \hline x & y & z \\ \hline \end{array} \end{array} = \left\{ a^{<\mathbf{r},A>}, b^{<\mathbf{r},B>}, c^{<\mathbf{r},C>}, x^{<\mathbf{s},A>}, y^{<\mathbf{s},B>}, z^{<\mathbf{s},C>} \right\} = \begin{array}{c} C \ B \ A \\ \mathbf{s} \begin{array}{|c|c|c|} \hline z & y & x \\ \hline \end{array} \\ \mathbf{r} \begin{array}{|c|c|c|} \hline c & b & a \\ \hline \end{array} \end{array}$$

Notice that the above expression is an equation asserting that the mathematical identity of both sides of the expression is the same. This means that the array-resembling object on left side and the array-resembling object on right side are the same mathematical objects, not different array objects as are RDM-tables. Notice, also, in this example our previous distinct RDM-tables, \mathbf{T}_1 and \mathbf{T}_{12} have identical RAM/DISK representations.

Q: But where do the \mathbf{r} and \mathbf{s} that label the rows come from?

A: These labels are system assigned, totally arbitrary, unique values, and are totally transparent to all operations defined for DLA operands. The system uses these hidden values as unique record identifiers for internal manipulation of physical data.

Q: I don't see any connection between DLAs and any physical data representations that I am familiar with.

A: Are you familiar with *Flat Files*?

Q: Of course.

A: Good. Let's do a flat file mapping of the above DLA. Keep in mind here that the ground rules for a mapping is to preserve the same set-membership on both sides of the I/O interface so that different operations on either side can produce the same resultant set-membership.

Q: That point has been well established.

A: Would you accept the following representation of a flat file and its associated mathematical identity?

$$\mathbf{FF1} = \begin{array}{|c|c|c|} \hline b & a & c \\ \hline y & x & z \\ \hline \end{array} = \left\{ \langle b, a, c \rangle^1, \langle y, x, z \rangle^2 \right\}$$

Q: Sure. I also accept that $\mathbf{FF1}$ is just one of twelve possible flat file choices. I assume you are about to describe a unique mapping operation that works for any of the twelve choices.

A: Quite correct. The following will be quite simplistic, but will demonstrate the central theme. For $\mathbf{Q} = \left\{ a^{<\mathbf{r},A>}, b^{<\mathbf{r},B>}, c^{<\mathbf{r},C>}, x^{<\mathbf{s},A>}, y^{<\mathbf{s},B>}, z^{<\mathbf{s},C>} \right\}$, I will show how two extended set processing (XSP) operations \mathbf{DtoP} and \mathbf{PtoD} provide reciprocal mappings between DLAs and RAM/DISK representations.

Q: I don't recall your using the term extended set processing, or XSP, operations before. What is the definition of an XSP operation.

A: Sorry, an XSP operation is any computer based operation that takes sets as operands, produces sets as results, and has a documented definition in terms of extended set theory (XST).

Q: So the XSP operations \mathbf{DtoP} and \mathbf{PtoD} , that you are about to use for an example, are documented in terms of XST?

A: Yes, but the arcane XSN is more disruptive to exposition than supportive, so let me just continue with a rather naive explanation. I will assert conditions using previously defined doubly labeled array **Q** and the flat file **FF1**:

$$\begin{aligned}\mathbf{FF1} &= \mathbf{DtoP}(\mathbf{Q}, \langle B, A, C \rangle, \langle \mathbf{r}, \mathbf{s} \rangle)^5 \\ \mathbf{Q} &= \mathbf{PtoD}(\mathbf{FF1}, \langle B, A, C \rangle, \langle \mathbf{r}, \mathbf{s} \rangle)\end{aligned}$$

Notice that these statements using XSP operations are both mathematical equivalences between sets and executable lines of code.

Q: Without going into the formal XST definition of these XSP operations would you just take one of them and show the transformation of input to result?

A: Sure. Let's take the first one, starting with the mathematical identity of **Q** and ending up with the mathematical identity of **FF1**, with $\mathbf{FF1} = \mathbf{DtoP}(\mathbf{Q}, \langle B, A, C \rangle, \langle \mathbf{r}, \mathbf{s} \rangle)$, giving:

$$1) \quad \mathbf{Q} = \left\{ a^{\langle \mathbf{r}, A \rangle}, b^{\langle \mathbf{r}, B \rangle}, c^{\langle \mathbf{r}, C \rangle}, x^{\langle \mathbf{s}, A \rangle}, y^{\langle \mathbf{s}, B \rangle}, z^{\langle \mathbf{s}, C \rangle} \right\}.$$

The third argument $\langle \mathbf{r}, \mathbf{s} \rangle$ indicates that all elements of **Q** need to be grouped by the first element of their scopes, giving:

$$2) \quad = \left\{ \left\{ a^{\langle \emptyset, A \rangle}, b^{\langle \emptyset, B \rangle}, c^{\langle \emptyset, C \rangle} \right\}^{\mathbf{r}}, \left\{ x^{\langle \emptyset, A \rangle}, y^{\langle \emptyset, B \rangle}, z^{\langle \emptyset, C \rangle} \right\}^{\mathbf{s}} \right\},$$

that the elements in $\langle \mathbf{r}, \mathbf{s} \rangle$ should be replaced by their scope values, giving:

$$3) \quad = \left\{ \left\{ a^{\langle \emptyset, A \rangle}, b^{\langle \emptyset, B \rangle}, c^{\langle \emptyset, C \rangle} \right\}^{\mathbf{1}}, \left\{ x^{\langle \emptyset, A \rangle}, y^{\langle \emptyset, B \rangle}, z^{\langle \emptyset, C \rangle} \right\}^{\mathbf{2}} \right\},$$

The second argument $\langle B, A, C \rangle$ indicates that all elements with scopes not containing an element of $\langle B, A, C \rangle$ should be eliminated, and that scopes that do should be replaced by the appropriate element of $\langle B, A, C \rangle$, giving:

$$4) \quad = \left\{ \left\{ a^A, b^B, c^C \right\}^{\mathbf{1}}, \left\{ x^A, y^B, z^C \right\}^{\mathbf{2}} \right\},$$

that the elements in $\langle B, A, C \rangle$ should be replaced by their scope values, giving:

$$5) \quad = \left\{ \left\{ a^2, b^1, c^3 \right\}^{\mathbf{1}}, \left\{ x^2, y^1, z^3 \right\}^{\mathbf{2}} \right\},$$

which is the mathematical identity of **FF1**,

$$6) \quad = \left\{ \langle b, a, c \rangle^{\mathbf{1}}, \langle y, x, z \rangle^{\mathbf{2}} \right\} = \mathbf{FF1}.$$

Q: That wasn't as painful as I thought it would be. Given that I accept the idea that a third alternative to I/O access process to be a mathematical transformation of data between RAM and DISK, how does that relate to improved performance through information density management?.

VI. SELECTIVE SET RETRIEVAL

A: By providing a third alternative to I/O access, *selective set retrieval* (SSR), that can actually improve I/O performance to 80% to 90% of optimal for information extraction applications.

Q: You're using the same definition of optimal that gave 2% for indexed record access? That's 40 times as fast! I'm prepared to believe a factor of ten, but forty times seems a little high.

A: I'll let you be the judge, but to give a fair judgment you will have to listen to some set theory.

⁵These XSP operations are somewhat stylized for this exposition. The actual XSP operations use functional descriptions for the second and third arguments instead of discrete sets.

Q: I can listen.

A: I assume that you are familiar with the set-theoretic distinction between a *partition* of a set and a *decomposition* of a set.

Q: Refresh my memory.

A: Both are collections of subsets of a given set such that the union of all the subsets equals the given set. The difference is that in a partition all the subsets are disjoint⁶ while in a decomposition two or more subsets may have a non-null intersection.

Q: I recall the distinction.

A: Good. Set partitioning and set decomposition are the methods used under set processing for isolating informationally dense subsets of available data to provide a minimal covering of necessary data at each step of an applications execution.

Q: Data partitioning is already a standard I/O performance strategy.

A: Dynamic physical data reorganization during application execution under mathematical control?

Q: Not usually.

A: The idea behind selective set retrieval supported buy set processing is really quite simple: the intelligent manipulation of subsets of physical data. To start let's look at a new DLA, **P**:

$$\mathbf{P} = \begin{array}{c} \begin{array}{c} \text{N S A M} \\ 1 \begin{array}{|c|c|c|c|} \hline b & 1 & 43 & d \\ \hline \end{array} \\ 2 \begin{array}{|c|c|c|c|} \hline e & 0 & 26 & s \\ \hline \end{array} \\ 3 \begin{array}{|c|c|c|c|} \hline h & 0 & 51 & m \\ \hline \end{array} \\ 4 \begin{array}{|c|c|c|c|} \hline g & 0 & 18 & s \\ \hline \end{array} \\ 5 \begin{array}{|c|c|c|c|} \hline a & 1 & 23 & s \\ \hline \end{array} \\ 6 \begin{array}{|c|c|c|c|} \hline k & 1 & 72 & m \\ \hline \end{array} \end{array} = \left\{ \begin{array}{l} b\langle 1, \mathbf{N} \rangle, 1\langle 1, \mathbf{S} \rangle, 43\langle 1, \mathbf{A} \rangle, d\langle 1, \mathbf{M} \rangle, e\langle 2, \mathbf{N} \rangle, 0\langle 2, \mathbf{S} \rangle, 26\langle 2, \mathbf{A} \rangle, s\langle 2, \mathbf{M} \rangle \\ h\langle 3, \mathbf{N} \rangle, 0\langle 3, \mathbf{S} \rangle, 51\langle 3, \mathbf{A} \rangle, m\langle 3, \mathbf{M} \rangle, g\langle 4, \mathbf{N} \rangle, 0\langle 4, \mathbf{S} \rangle, 18\langle 4, \mathbf{A} \rangle, s\langle 4, \mathbf{M} \rangle \\ a\langle 5, \mathbf{N} \rangle, 1\langle 5, \mathbf{S} \rangle, 23\langle 5, \mathbf{A} \rangle, s\langle 5, \mathbf{M} \rangle, k\langle 6, \mathbf{N} \rangle, 1\langle 6, \mathbf{S} \rangle, 72\langle 6, \mathbf{A} \rangle, m\langle 6, \mathbf{M} \rangle \end{array} \right\}$$

There are 24 data elements in set **P**. One partitioning of **P** would be in six groups of four to represent individual rows of an array. Another partitioning of **P** would be in four groups of six to represent individual columns of an array. Okay so far?

Q: So far, okay, but could I make a suggestion?

A: Sure.

Q: For exposition purposes, it might help if you could assign some meaning to the elements of **P**.

A: How about people (**P**) with names (**N**), sex (**S**), age (**A**), and marital status (**M**)?

Q: That'll work.

A: Remember that the row-labels are totally arbitrary as long as they are unique. Numbers were chosen strictly for conceptual convenience and give no clue to how the physical data is organized.

Q: You made that quite clear earlier.

A: Good. So let's see what the idea of a minimal query covering buys us. To get the spirit of selective subset retrieval I/O access we will need some subsets that can be combined to form query coverings. Next we need to combine these subset to provide minimal coverings for specific queries.

⁶The intersection of any two sets is empty.

Given that \mathbf{P} is a DLA, subsets of \mathbf{P} will also be DLAs. Three such subsets are defined as follows:

$$\mathbf{M} = \begin{array}{c} \mathbf{N} \ \mathbf{S} \ \mathbf{A} \ \mathbf{M} \\ \begin{array}{|c|c|c|c|} \hline h & 0 & 51 & m \\ \hline k & 1 & 72 & m \\ \hline \end{array} \\ \begin{array}{l} 3 \\ 6 \end{array} \end{array} = \left\{ h^{<3,N>}, 0^{<3,S>}, 51^{<3,A>}, m^{<3,M>}, k^{<6,N>}, 1^{<6,S>}, 72^{<6,A>}, m^{<6,M>} \right\}$$

$$\mathbf{S} = \begin{array}{c} \mathbf{N} \ \mathbf{S} \ \mathbf{A} \ \mathbf{M} \\ \begin{array}{|c|c|c|c|} \hline e & 0 & 26 & s \\ \hline g & 0 & 18 & s \\ \hline a & 1 & 23 & s \\ \hline \end{array} \\ \begin{array}{l} 2 \\ 4 \\ 5 \end{array} \end{array} = \left\{ e^{<2,N>}, 0^{<2,S>}, 26^{<2,A>}, s^{<2,M>}, g^{<4,N>}, 0^{<4,S>}, 18^{<4,A>}, s^{<4,M>}, a^{<5,N>}, 1^{<5,S>}, 23^{<5,A>}, s^{<5,M>} \right\}$$

$$\mathbf{D} = \begin{array}{c} \mathbf{N} \ \mathbf{S} \ \mathbf{A} \ \mathbf{M} \\ \begin{array}{|c|c|c|c|} \hline b & 1 & 43 & d \\ \hline \end{array} \\ 1 \end{array} = \left\{ b^{<1,N>}, 1^{<1,S>}, 43^{<1,A>}, d^{<1,M>} \right\}$$

I am assuming from your obvious knowledge of set theoretic concepts, that you recognize that subsets \mathbf{M} , \mathbf{S} , and \mathbf{D} as a partition of \mathbf{P} .

Q: Yes, I see that the union of \mathbf{M} , \mathbf{S} , and \mathbf{D} equals \mathbf{P} . I also see that if an application is oriented toward marital status: only a third is required if married people are of interest, one half is required if single people are of interest, and only one sixth is required if divorced people are of interest. Are these partitions examples of minimal coverings?

A: Yes, but only of the simplest kind. The real strength of minimal coverings from a performance perspective is in what data can be ignored. Dr. F. H. Westervelt⁷ once remarked that “You can’t do faster than which you don’t have to do, then by not doing it”. The point applied here is that applications oriented toward marital status can ignore two thirds of the data for interests in married people, half the data for interests in single people, and a full five sixths for divorced people.

Q: Seems like just another way of looking at a glass as being half empty or half full.

A: I think the choice depends of whether you are trying to fill the glass or empty it. In this case performance depends on how big the glass is, how full it was to start, and how little is left in it. As you have probably already experienced, if you have an application that requires access to a Gigabyte of data and total size of your data is a Gigabyte you will experience much better performance than if that Gigabyte of data were strewn throughout a terabyte of irrelevant data.

Q: So the rationale behind selective set retrieval is to be able to populate storage with enough sets with enough variety that for any application just that combination of sets that can contribute to the result are recognized.

A: Absolutely correct. Let’s further explore our simple example, then move on to one from an industry standard. Our three subsets of \mathbf{P} were *subsetsed horizontally*, or row-wise. \mathbf{P} can also be *subsetsed vertically*. \mathbf{P} could already be a 4-domain vertical partition of a 20-domain original DLA.

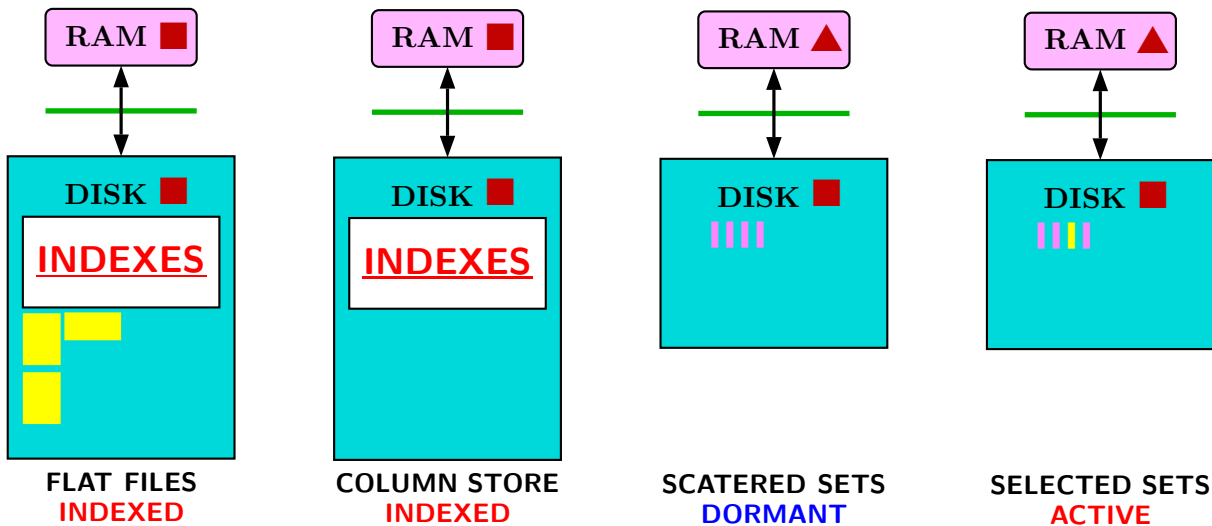
Q: That would make the mythical glass 5 times bigger, which means our example can ignore 14/15 of the data for interests in married people, 9/10 of the data for interests in single people, and 29/30 of the data for divorced people.

A: Quite true, and this brings us back to the concept of using the information density of I/O transfers as measure of performance.

Q: But only for information extraction applications, since transaction processing applications are informationally dense in that they require all the fields of a record.

⁷Director of CONCOMP, an ARPA research project, who in 1965 initiated the investigation into exploring whether data structures could be treated as mathematical objects. It resulted in the development of extended set theory.

A: Quite correct. We are concerned with the performance of accessing many, many records where just a fraction of each record needs to be present. This is precisely why selective set retrieval gives such dramatic performance advantages over indexed record access. Consider the following distorted I/O interface visual abstractions:



Q: Why distorted?

A: Actually I was being flippant, but I was trying to indicated that the indexed structures really take up much more storage⁸ than is indicated by the visual aid.

Q: I heard what you have been saying and it makes sense as far as you have gone, but so far it has all been about set modeling and nothing about practical implementation concerns, like: load times, storage requirements, updates, ACID requirements, distributed data access, XML documents, and much more.

A: Quite true, but in defense of introducing a radical (possibly hostile and certainly iconoclastic) concept, I wanted to establish some building blocks that allow us a common ground for comparisons.

Q: Fair enough, but I do anticipate an eventual consideration of my implementation concerns.

A: They're in the queue.

VIII. FUTURE SYSTEMS

Q: The performance difference between Indexed Data Access I/O and Selective Set Retrieval I/O for information extraction applications has been validated many times. Would it not be to a company's competitive advantage to provide customers the ability to access information dramatically faster?

A: It would certainly seem like good business judgment.

Q: Don't you believe it is just a matter of time before some enterprising company recognizes that XSP technology could do for systems modeling and development today what the RDM did for the advance of systems modeling and development some thirty odd years ago?

A: Possibly, but let me ask you a question. Given what you now know about the complexities of

⁸Typically 10 to 70 times the amount of storage required for the raw data. Look at the load requirements for the TPC-H benchmark [?].

modeling data, what would you, as a systems architect, use?

Q: Why not sets?

IX. EPILOGUE

HAROLD⁹: This preceding hypothetical dialogue was constructed in an attempt to expose an available and proven technology that defies ready acceptance by those who could exploit it best.

References

- [BG] Blass, A; Gurevich Y.: *Why Sets?*, Bull. Eur. Assoc. Theor. Comput. Sci. 84 (October 2004)
<http://research.microsoft.com/~gurevich/Opera/172.pdf>
- [Bo75] Boyce, R. F.; Chamberlin, D. D.; King, F. W.; Hammer, M. M.: *Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage*, Comm. of ACM, Vol. 18, No. 11, 1975.
- [Carr] Carroll, L.: *Through the Looking-Glass*, Ch.4, 1872
- [Ch68] Childs, D. L.: *Feasibility of a Set-Theoretic Data Structure: A General Structure Based on a Reconstituted Definition of Relation*, Proc. IFIP Congress 1968
- [Ch77] Childs, D. L.: *Extended Set Theory: A General Model for Very Large, Distributed, Backend Information Systems*, Third International Conference On Very Large Databases, Tokyo, Japan, 1977
http://xsp.xegegis.org/VLDB_77abstract.pdf
- [Ch86] Childs, D. L.: *A Mathematical Foundation For Systems Development*, NATO ASI Series, Vol F24, Database Machines, Edited by A. K. Sood and A. H. Qureshi, Springer-Verlag, 1986
http://xsp.xegegis.org/Nato_asi.pdf
- [Ch95] Childs, D. L.: *Axiomatic Extended Set Theory*, 1995
- [Co70] Codd, E. F.: *A Relational Model of Data for Large Shared Data Banks*, CACM 13, No. 6 (June) 1970, [p. 379-380]
<http://www.cs.nott.ac.uk/~nza/G51DBS/codd.pdf>
- [Ma83] Maier, D.: *The Theory of Relational Databases*, Computer Science Press, 1983, [p. 373]
- [Mc60] McCarthy, J. L.: *Recursive Functions of Symbolic Expressions and Their Computation by Machine*, Communications of the ACM, 3(4), 184-95, 1960.
- [Sk57] Skolem, Thoralf: *Two Remarks on Set Theory*, Mathematica Scandinavica 5 (1957), p.43-46.
<http://www.msccand.dk/article.php?id=1481>
- [Su60] Suppes, Patrick: *Axiomatic Set Theory*, Van Nostrand, 1960, p.20 & 141.
- [TPC] TPC-H Specifications
<http://www.tpc.org/tpch/spec/tpch2.7.0.pdf>

⁹From a book by the Nobel-prize-winning physicist Julian Schwinger in which the exposition is occasionally interrupted by ‘HAROLD’, an acronym for the ‘hypothetical alert reader of limitless diligence’.