ARMY RESEARCH LABORATORY

Outer Analysis of Quality

by Trevor Cook

ARL-TR-7159 January 2015

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Adelphi, MD, 20783-1138

ARL-TR-7159 **January 2015**

Outer Analysis of Quality

Trevor Cook

Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2015	Final	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Outer Analysis of Quality	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)	5d. PROJECT NUMBER
Trevor Cook	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
US Army Research Laboratory ATTN: RDRL-CIN-T 2800 Powder Mill Rd. Adelphi, MD 20783-1138	ARL-TR-7159

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We use type signatures to discuss several specific notions of quality in order to showcase how the simple discipline can aid in the inspection and definition of vague and intuitive notions. We model qualities as functions, thereby shifting attention from what they are to what they may be based upon. The ultimate aim being the design of systems that can deliver high quality data for arbitrary definitions of quality. The report includes a short description of algebraic data types.

15. SUBJECT TERMS

Quality of Information, Category Theory, Type Theory

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Trevor Cook
Unclassified	Unclassified	Unclassified	UU	28	19b. TELEPHONE NUMBER (*Include area code*) (301)-394-1851

Contents

1. Introduction **1**

2. Qualities and Ordering **2**

3. Data and Information **2**

4. Pertinence **4**

5. Completeness **5**

6. Accuracy **7**

7. Precision **8**

8. Freshness **10**

9. Timeliness **12**

10. Estimation **12**

11. Related Work **14**

12. Conclusion **15**

13. References and Notes **16**

Appendix. Algebraic Data Types **17**

Distribution List **21**

1. Introduction

In this report, we analyze some common concepts of data quality. Indirectly, our aim is to remove ambiguities associated with these concepts, with the ultimate goal of designing systems that deliver high quality data. Our immediate aim, however, is to provide an alternate approach to thinking about–and communicating notions of–information and quality.

If we want to specify general, universally applicable definitions of various qualities, we must avoid specifying how to calculate any particular quality relating to specific data and applications. One approach to this is to propose an ontology that captures the essential structure of quality and then somehow decide how any specific metric fits with the ontology. This approach is not undertaken here. We leave this structure open and instead focus on giving form to individual notions–without regard to whether or not they are necessarily parts of a specific collection, "the qualities."

In this report, we present an "outer analysis" of qualities, a circumscription of each particular notion. Outer analysis is a term associated with category theory. Its connotation is an investigation of (mathematical) objects by looking at their relationships with other objects, as opposed to inspecting the internal, element-wise structure. In this vein, we describe qualities such as accuracy, etc., as *functions* and analyze them in terms of their inputs and outputs.

We use type signatures with forms similar to $\lambda : \mathbb{Q} \to \mathbb{N}$. This example signature describes some function named λ, which takes rational numbers, $\mathbb{Q}$, in the domain to natural numbers, $\mathbb{N}$, in the co-domain. For example, the ceiling of the absolute value of a rational, $\lambda\left(q\right) = \lceil |q| \rceil$, adheres to this type signature.

Type signatures are used in category theory, form the basis of various type theories (including typed lambda calculi), and are often generally used in the presentation of various branches of mathematics. This analysis is not a rigorous treatment based solely in any one discipline, however. But we do want to hint at least that such a treatment is based in very deep and foundational theories of math and logic, and more rigorous treatments will benefit from the insights discovered in those theories.

It is likely that most readers will disagree with at least some parts of the descriptions of the qualities outlined below. Let me stress that we make no claim that these descriptions are either authoritative or comprehensive. I do not even claim that the treatment is entirely consistent.

Where flaws are found, please consider the role of the signatures in finding those inadequacies.

2. Qualities and Ordering

Whether implicit or explicit, we say that a particular need for information relates to an evaluation of data encapsulated by a function with the form

$$quality : D \to Q,$$

where D is the type of data and Q is at least a partially ordered set. So $quality$ provides an ordering or preference by assigning each value $d : D$ (pronounced d of type D) to value $q : Q$. The data, D, can be a complex compound consisting of multiple sub-types and/or choices between types of sub-data, especially as an algebraic data type (ADT).[1]

A partial ordering, or poset, is a set equipped with an ordering relation, $\leq$. The ordering is reflexive, $q \leq q$, transitive, i.e., $q \leq q', q' \leq q'' \implies q \leq q''$, and antisymmetric, $q \leq q', q' \leq q \implies q = q'$. The ordering is partial though; for any $q, q' : Q$, it may be the case that neither $q \leq q'$ or $q' \leq q$. The need for a preference over data requires some sort of ordering, and posets are nearly the weakest kind of ordering, so we take Q (and variants: $Q', Q^*, \ldots$) to be a poset in this report.[2]

3. Data and Information

We use 2 objects–D and I–in the development of specific descriptions of qualities. Both objects should be taken as a variable of sorts, where D is any object we would consider as data and I can be any object we might call information. We don't say much about the particular nature of D or I; it's mostly just an assertion that there are two different concepts. The reason we differentiate at all arises from the definition of $accuracy$, as a comparison between the captured data, D, and some underlying thing the data represents, I. This choice also informs other decisions, for instance, defining $pertinence$ in terms of I and $precision$ in terms of D.

Although it is perhaps unnecessary in this treatment, we can give a little more of the working intuition about the relationship between D and I. We say that the data, D, indicates, contains–or may indicate–the information, I, and that we acquire data for sake of the information it contains.

The information component of data is a partial function, represented as mapping from data to either the underlying information, I, or a constant "no information" value, $\mathbf{1}$:

$$info : D \to I + \mathbf{1}.$$

Given a piece of data, $d : D$, $info\,(d)$ might yield either a piece of information of the requisite type, $i : I$, wherever the partial function is defined, or a default "unknown" value, $1 : \mathbf{1}$, for undefined inputs. The details for why the form of $info$ is equivalent to a partial function over D are given in Awodey,[3] example 7.26.

As an example of the above, a photograph may provide information about a person's location, written

$$info_{look} : Image \to (Person \times Location) + 1,$$

where the information in this case is the pair of objects $Person$ and $Location$. Given an image, we might learn something about a person being at a location, e.g.,
$info_{look}\,(pic) \mapsto$ ("Al", "Times Square"). On the other hand, if the image was blank we get, $info_{look}\,(blank) \mapsto 1$.

The signature of $info$ also provides a default method for injecting any data to the realm of information, with, $info_{pure}\,(d) \mapsto d$. In that case, the data itself becomes the object of interest–we want the object for the object's sake. Such was the case when I was looking for the picture of the time I met Al Pacino in Times Square.[4] I already knew the contents of the picture, nonetheless I still wanted it.

We also hold that the reverse case need not be possible. That not all information can be injected into the realm of data. For instance, $Location$ as an instance of information, may be used to represent actual physical places, not just coordinates or names. We may, for instance, derive from a location coordinate data, but the location itself might be unrepresentable.

I find the above distinctions between D and I intuitively pleasing. However, we shouldn't insist that these distinction are *the* right way to look at these concepts, or even necessary–in the sequel or in general. The development of these intuitions were done in tandem with the resulting quality descriptions, so any rethinking of the below may also require some reworking of the above.

4. Pertinence

Pertinence is the notion of having the "right" information, in the sense that a piece of information, $i : I$, may or may not be topical. As a result, we might say pertinence evaluations have the following form:

$$pertinence : I \to \mathbf{2},$$

where $\mathbf{2}$ is the set $\{0, 1\}$ with $0 \leq 1$. This follows the strict true/false concept of pertinence. We define pertinence over, I, with the expectation that I is *the* thing we want to know. Generalizing this, we can alternatively say

$$pertinence : I \to Q$$

if information has varying levels of applicability.

As an example of pertinence, if I want to know the temperature outside, I may have a function,

$$pertinence_{loc} : Location \times Temp \to Q,$$

which assigns a higher evaluation to locations near my window than in the next town over. So, my window thermometer may be more pertinent than a reading I get from a weather station via the Internet. From the discussion above, we take $Location$ to be the actual location of the temperature reading and not some possibly flawed reporting.

To evaluate data in terms of pertinence, we want something of the form

$$pertinence_{data} : D \to Q.$$

We attempt to build a definition through the composition of pertinence with the function that gets the needed information from the data:

$$pertinence_{data} = pertinence \circ info. \tag{1}$$

However, the types do not match up. The output of $info$ is $I + 1$ and the input of $pertinence$ is I. We need to do something with the case where the information is not contained in the data.

The most general solution lies in the "lifting" of $I \to Q$ into context $I + 1 \to Q + 1$. This is available (since $+$ is functorial, and there is only ever one function to the terminal object 1,

$! : a \to 1$). The lifted function is written

$$pertinence' : I + 1 \to Q + 1$$

$$pertinence' = pertinence + !. \tag{2}$$

The resulting definition can be given with

$$pertinence_{data} : D \to Q + 1$$

$$pertinence_{data} = (pertinence + !) \circ info. \tag{3}$$

If we let Q^* be a pointed set with a distinguished element identified by $* : 1 \to Q^*$, we can collapse $Q + 1$ to Q^*, via the coproduct $[id, *]$. Thus we achieve a function from D to Q^* with

$$pertinence_{data} : D \to Q^*$$

$$pertinence_{data} = [id, *] \circ (pertinence + !) \circ info. \tag{4}$$

The above development may be daunting, so some exposition is in order. There are 3 entities involved in the evaluation of data's pertinence. First, we need a way, $info$, to map the data to the underlying information that we are interested in, acknowledging that the information may not actually be there. Second, we need to define a way, $pertinence$, of saying how topical the information is, without regard to other quality questions. Third, we need to assign a quality to the case wherein the information was not in the data, $*$. Besides those 3 elements, the other aspects, id, $[\ ,\ ]$, $+!$, exist as invariants and plumbing for the functional composition.

---

5. Completeness

We describe completeness as the totality of pertinent information. Like pertinence, we also take completeness to be an evaluation solely based on information,

$$completeness : I \to Q,$$

with the understanding that some information will tell a more complete story than others. As it stands, completeness and pertinence are indistinguishable at the type level, so this description is unhelpful.

We can distinguish them by imposing some additional structure on the information's type, I, through the use of parametric polymorphism. We let $S\ p$ be some structure, S, inhabited by any underlying primitive type, p. Some examples of such structures are powersets, $S\ p = 2^p$; pairs or other n-tuples, $S\ p = p \times p$; homogeneous lists with all elements of type p, $S\ p = [p]$; or even structures additionally inhabited by concrete types, $S\ p = ErrorCondition + p$, or $S\ p = \mathbb{N} \times p$. Note that the foregoing are all examples of algebraic data types.

Following the above, we let information be defined as a structure over primitives, $I = S\ P$, completeness be defined as some evaluation over information,

$$completeness : S\ P \to Q',$$

and let pertinence be an evaluation over the primitives,

$$pertinence : P \to Q.$$

This leads to the possibility of using $pertinence$ in the calculation of $completeness$. As such, we have a signature that takes a pertinence function as an input:

$$completeness : (P \to Q) \times S\ P \to Q'.$$

Notice that given a particular pertinence, $p : P \to Q$, the signature resolves to the previous signature, $completeness : S\ P \to Q'$.

As a concrete example, we can build a completeness measure from pertinence based on subset membership. Assuming that we want $completeness\,(s_1) \leq_{Q'} completeness\,(s_2)$ only when the pertinent subset of s_1 belongs to that of s_2. We merely have to define $Q' = 2^P$, with the ordering, $\leq_{Q'}$, inherited via the subset relationship of 2^P. Next, we use a particular pertinence evaluation, p, to find the subset of pertinent information with

$$completeness_{pertMemb} : (P \to \mathbf{2}) \times S\ P \to S\ P$$

$$completeness_{pertMemb}\,(p, s) = \{i | i \in s, p\,(i) = 1\}\,. \tag{5}$$

Of course, this definition of completeness may not suit some tasks. It only allows comparison between sets of information when one is a subset of the other. Instead, a task might demand that

the greatest quantity of pertinent information is favored. In that case, we may have $Q' = \mathbb{Z}$, and

$$completeness_{sum} : (I \to \mathbb{Z}) \times S\ I \to \mathbb{Z}$$

$$completeness_{sum}\left(p, s\right) = \sum_{i \in s} p\left(i\right). \tag{6}$$

Different behavior can now be obtained with the definition of pertinence. For instance, $p\left(i\right) \mapsto \{0, 1\}$ treats all equally sized subsets of pertinent information as the same completeness, $p\left(i\right) \mapsto \{-1, 1\}$ will favor subsets with only pertinent information over subsets with non-pertinent information, and of course, any general $p\left(i\right) \mapsto n \in \mathbb{Z}$ is still available. Going further still, we need not tie completeness to summations of pertinence; we could have instead used products, logical expressions, or any other operation that the types allow.

Finding the completeness of D, as opposed to I, follows the same development as $pertinence$, above. Namely, for a given completeness measure, we have to decide how the data maps to the info, $info$, and how to rate the null information, $*$.

---

6. Accuracy

Our notional concept of accuracy is that it measures how well the data reflect the intrinsic information. We write $accuracy : I \times D \to Q$, as any function that somehow compares data, D, to the underlying information, I. This is the first time we have used the concepts of both data and information together, and it is indeed accuracy that makes the separation necessary. An example of accuracy is the image compression quality metric, picture signal-to-noise ratio (PSNR), which compares a compressed image with that of the original, $psnr : Image \times Image \to \mathbb{R}$.

Of course, there are accuracy concerns waiting in the wings of the above quality considerations, such as $pertinence$. As the signature requires, the concern rises through the use of the function that invokes both objects; $info : D \to I + 1$. In the case of $pertinence$ of location data, for instance, we want our data to be proximal to a location, but we also want our reported location to reflect our actual location. We are careful to provide a separation between these two concerns, which is enabled to a large part through our distinction of D and I.

Accuracy is a fundamentally important concern in the analysis of quality. It comes into play whenever we capture data from information or when we project data into the realm of information. Assuming that all data arises from information, we might measure accuracy solely in

terms of information

$$accuracy : I \times I \to Q,$$

where one I is the original information and the other I is the inferred information. Following this, any data calculation for accuracy would depend on both the method in which that information was captured, $I \to D$, and the method by which it was injected back into information, $D \to I$:

$$accuracy : (I \to D) \times (D \to I) \times D \to Q.$$

We develop a modest treatment of accuracy in our dealings with $freshness$; however, a full treatment is beyond the scope of this report.

---

7. Precision

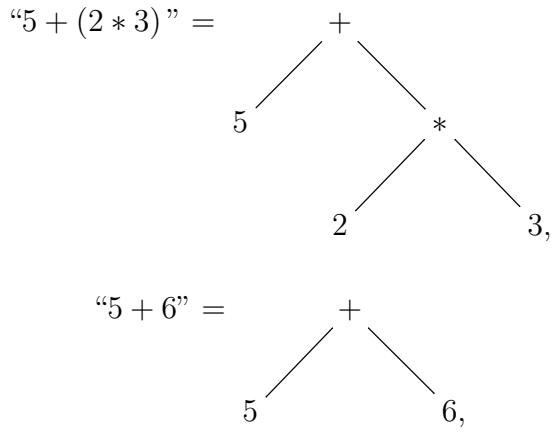
Precision is a measure of the ability to capture or express information. In empirical sciences, it relates to the repeatability of measurement and is also often described as the significant digits of a number. Given the deliberate divorce of precision from "actual" value, we take precision to be an evaluation based purely on the structure of the data,

$$precision : D \to Q.$$

It is unclear how to add any clarity to the concept with this definition of precision. As it stands, any function that inspects data and returns a number fits this signature. Also, the definition in terms of significant digits is already a good one. One area of particular interest not covered by significant digits, however, is in the precision of language.

Consider simple algebraic strings using numerals, variables, addition, multiplication, etc., as an example of a language whose statements we want to evaluate. Some example expressions in the language are "$5 + (2 * 3)$", "$5 + 6$", and "11". All of these strings represent the same thing, 11. However, more information is contained in "$5 + (2 * 3)$" in that it tells one more about how the eventual value is produced.

If we look at the expressions as abstract syntax trees, we get

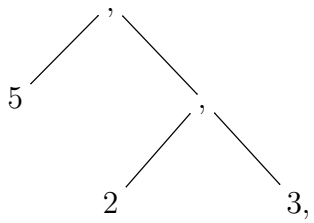


and

$$\text{“11”} = 11\,.$$

This helps illustrate the point that there is an underlying structure to our expression as well as the data present at each node in the structure.

We shouldn't jump to the conclusion that a bigger structure is more precise, however. A list $[5, 2, 3]$ can be represented with a structure such as



but we can succinctly specify a list of 100 elements, each with value 1, without writing down the whole 99-level-deep structure. So if our language is strong enough to specify both lists and the shorthand for creating lists, comparing statements may entail program calculation. This brings to bear questions of reduction strategies.

With formal languages, different reduction strategies can lead to differences in the result, especially in the case of termination vs. non-termination. So we have the possibility of ambiguity in language through the choice of calculation. Natural languages are worse still, in that statements are often open to interpretation.

Ambiguity is a key element of precision, as we consider a statement to be more precise when there are fewer possible interpretations. We must be careful, though; by invoking interpretation,

we are letting the "actual" meaning, i.e., the information, creep back into our definition.

8. Freshness

Freshness is a quality metric where we rate the data based on how old it is. Unlike the other metrics, this is the first instance of asserting a specific data/information entity, $Time$. For freshness, we need 2 instances of time, the data's time and the current time,

$$freshness : Time \times Time \to Q.$$

We choose the pair, $Time \times Time$, as opposed to some time difference, $\Delta Time$, because there is more information with the pair. This signature shows freshness as a preference over a pair of times, or, given the current time, a preference over age. The signature is nice in its simplicity but, for better or worse, divorced from the data whose freshness is being evaluated.

We could add some detail to this by specifying how exactly the time is related to the data and how the current time is derived. Indeed, we need some functions,

$$\pi_{time} : D \to Time$$

$$clock : World \to Time,$$

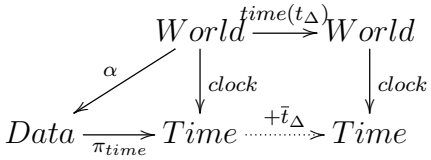
which get the time from the data and the time from the world. Then, passing these functions to freshness, we have the signature

$$\begin{array}{rccc} freshness: & (D \to Time) & \times & (World \to Time) \\ & & \times & (World \times D) \\ & & \to & Q, \end{array}$$

which makes explicit that freshness depends not only on an evaluation of $Time \times Time$, but on how we came to know this pair of times.

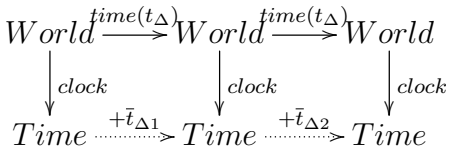
Just for fun, I elaborate further. We really want freshness to reflect an actual passing of time. That is, our data's time and our current clock's time should both reflect a function, $time\,(t_\Delta)$, which

ages the world t_Δ units, as in the following diagram:

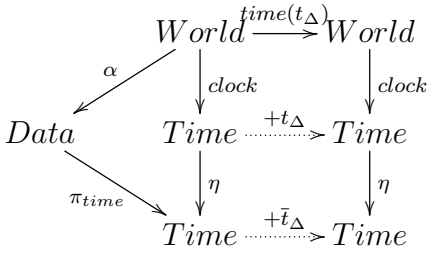


As such, our calculation entails estimating the parameter, Δt, such that all paths in the diagram are equivalent (the diagram commutes). However, there are a few wrinkles to this.

First, we don't know whether or not $clock$ captures time uniformly. Specifically, given 2 equivalent durations of time, t_Δ, we have no assurance that our 2 estimations, $\bar{t}_{\Delta 1}$, $\bar{t}_{\Delta 2}$, are equal:



This can be modeled by assuming a perfect clock, in the sense that any $\bar{t}_{\Delta 1}$, $\bar{t}_{\Delta 2}$ as above both equal t_Δ. Then a function, η, adds error to the reading as a function of time:



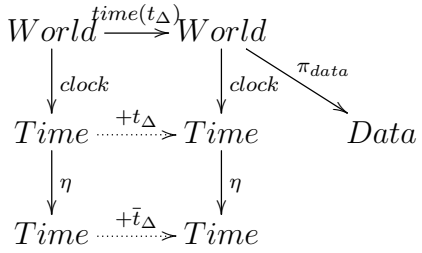
Now, freshness can be seen not only as a function of a pair of times, but also of an error function, η, whose value at the data's capture time is determined in part by how the data are captured, α, and whose value at the current time is determined by the implementation of a current clock.

From the above, we see that the calculation of freshness isn't pretty, in that, it carries with it further concerns about quality. Specifically, there is an accuracy concern of the form, $accuracy_{fresh} : (Time \times Time) \times (Time \times Time) \rightarrow Q$, wherein we compare the times read by a perfect clock versus the times read by the user. I have illustrated the concern here for the sake of narrative, but I don't mean to imply that it is a characteristic of freshness alone. Indeed, any quality measurement may carry with it associated accuracy concerns.

9. Timeliness

To round off the quality analyses, we visit timeliness, which reflects the notion that the usefulness of data depends on when it is inspected. Like freshness, timeliness can be specified with $timeliness : Time \times Time \rightarrow Q$. Here however, we assume that the times now represent the current time and the future time. Like freshness, the signature is nice in its simplicity; it is simply a preference over the future. Again though, for better or worse, it is divorced from the data whose timeliness it evaluates.

The elaboration for this scenario mirrors that of freshness, with the exception that we no longer rely on the data to tell us the time it was captured:



This diagram shows that after the world advances some t_Δ from its current state, we can check the clock and also inspect the data, π_{data}. Any disparities, η, from the perfect clock are now a function of the actual time that was inspected during those two instances.

This is a simple scenario. Worse yet, it seems to force one to wait around t_Δ to make a calculation. We hold that this is how timeliness should be specified, as a preference over the current time and future times. However, we want to make the judgment of whether or not we should attempt to access the data, so to continue our analysis, we look at estimating timeliness.

10. Estimation

Estimation of quality is fundamentally important to the task of providing high quality data. Consider the above discussion on timeliness, wherein a user needing data cares how long it takes for the data to be presented. Given that the user has presented the system with a trade-off function,

$$q : Z \times Time \rightarrow Q,$$

the system would be well advised to consider which forms of Z it may deliver at which $Time$ and select the estimate with the greatest evaluation.

Notice here that we intend to estimate quality based on an estimate of data. This is a tricky prospect, for we may have a process, $\lambda : Z' \times Time \to Z \times Time$, which transforms some Z' into the requisite type, Z, but also advances time, $Time$. However to evaluate, q, we need to provide a Z, which we will not have until running λ on the candidate data.

One enticing solution to this problem is by providing a general purpose data profiler,

$$prof : P_{(D' \to D)} \times (D' \to D) \times D' \to \left(D, P_{(D' \to D)}\right),$$

which for any $\alpha : D' \to D$, and initial profile of α, $P_{(D' \to D)}$, and input data $d' : D'$, we apply $\alpha\left(d'\right)$ to provide the output $d : D$ along with an updated profile. Next, when an estimate is needed, we use the profile and input data,

$$est : P_{(D' \to D)} \times D' \to D.$$

To perform the estimation of $(z', t) : Z' \times Time$, we would have the profile, $p_\lambda : P_{(Z' \times Time)}$, and would just call $est\left(p_\lambda, (z', t)\right)$, to yield (z, t). Such profilers do exist. Trivially, the profiler can essentially capture an exemplar $d : D$ and the estimate presents d for every subsequent estimation. Obviously, such an estimation scheme will often deliver poor results. Future research will consider ways one might beat the trivial profiler estimation.

As simplistic as the trivial profiler may be, it has the advantage of a constant evaluation time. This is an important point as our evaluation function, q, in this section considers time a factor. We haven't taken into account the time of estimation or the time taken by the functionality that is figuring what data to reply, but it should be apparent that the quality of the delivered data will depend on the timeliness of the estimation and the estimates they provide.

Another interesting nuance to this problem is in the self-referential estimation of the performance of the system when processing requests. The user gets the system to deliver data by supplying a quality evaluation to a request function, which we can pose as

$$request : St \times (D \to Q) \to (St \to D) \times St.$$

Given a current state, $st : St$, and the evaluation $q : D \to Q$, $request\left(st, q\right)$ returns a new state and a function allowing the user to access the data from the state, $\pi_D : St \to D$. We pose the

returned data in terms of the state to make specific the connection between the state and the required data. That is, we advance the state and then access the data.

If we let $app\,(f, x) = f(x)$, then we have a function, $app \circ request : St \times (D \to Q) \to D$, which returns a desired data based on some other data. As in the discussion above, the request function should consider functions that may yield desirable data and so should estimate its own expected performance. This bit of nuance is unsurprising and apparently relates to defining exit conditions on searches or tolerances on numerical estimations. Although unsurprising, it is satisfying to see this concern surface as a result of our analysis.

11. Related Work

The jumping off point of this report has its origin with Bisdikian[5] and similar publications.[6–9] The theme of these is in providing data in military networks (especially sensor networks) with quality metrics, annotation schemes, and a quality specification framework. The work may be suitable for closed and controlled networks; however, it is unclear whether or not the concepts are directly generalizable to arbitrary information systems.

One assumption in the above work is that various quality metrics can be captured at the source for later data consumers to base their usability judgments upon. The generalization leads to questions. For instance, can we capture an accuracy metric for all arbitrary data? Are the accuracy measurements of any two types of data directly related? If so, how do we ensure such? If not, in what way do they both capture the same thing?

To address those open questions, work similar to the approach taken in this report has been published[10, 11] where we argue that besides the data, provenance will be the basis for any quality evaluations. In other words, a preference over data quality requires a preference over the methods of creating data. The simple reason being the functions that create the data also determine the characteristics of the data. The prescription given by these works are accordingly simple–to provide high quality data, we should track data *and* functionality. This solution is elegant, since keeping track of functions is already needed to manipulate data for increased usability.

This report is similar to the work on provenance in that that work uses the same underlying language of type signatures. Those articles steer clear of putting to form any particular qualities, and so this report adds support for the applicability of those treatments.

12. Conclusion

In this report, we have explored several notions of quality through the use of type signatures. The purpose was to show how such a simple methodology can aid discussion of the concepts involved and will hopefully lead to design principles for quality aware systems.

I have been careful to mention that I do not claim these concepts to be the "true" or "right" concepts of quality. Accordingly, I make no claims that a comprehensive ontology of quality *exists*, or that *any* particular concept is applicable across all data. Certainly one should be skeptical of such claims without evidence. Notwithstanding, we take as given that data is variably suited to various applications, so something such as the foregoing analysis is in order.

With all our caveats, the question becomes, how does this discussion aid in the design of quality aware systems? Our primary contribution is in the reframing of the problem. This is a subtle shift, but instead of declaring that there are several quality dimensions that any data can be evaluated against, we leave the evaluations open ended and look at what these evaluations could possibly be based on. We have abandoned the prescriptive approach of declaring what quality *is* to one wherein we seek to support any evaluation of quality that may be provided.

13. References and Notes

1. See the Appendix for an introduction to ADT.

2. We could weaken from a poset to a preorder by dropping the antisymmetry requirement, but this doesn't appear to be useful.

3. Awodey S. Category theory. volume 49 New York: Oxford University Press; 2006.

4. Never actually happened.

5. Bisdikian C, Kaplan L, Srivastava M, Thornley D, Verma D, Young R. Building principles for a quality of information specification for sensor information. In: Information Fusion, 2009. FUSION '09. 12th International Conference on; p. 1370 –1377.

6. Bisdikian C, Verma D, Kaplan L, Srivastava M, Thornley D. Defining quality of information and metadata for sensor-originating information. In: 4th USMA Network Science Workshop; p. 1–14.

7. Bisdikian C, Kaplan LM, Srivastava MB. On the quality and value of information in sensor networks. ACM Trans. Sen. Netw. 2013;9(4):48:1–48:26.

8. Bar-Noy A, Cirincione G, Govindan R, Krishnamurthy S, LaPorta T, Mohapatra P, Neely M, Yener A. Quality-of-information aware networking for tactical military networks. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on; p. 2 –7.

9. Bisdikian C, Branch J, Leung K, Young R. A letter soup for the quality of information in sensor networks. In: Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on; p. 1–6.

10. Cook T, Toth A. Information processing and quality evaluations. In: Pervasive Computing and Communications Workshops (PERCOM Workshops); p. 1–7.

11. Cook T, Scott L. Type based abstraction for qoi aware applications. In: Military Communications Conference, 2012 - MILCOM 2012; p. 1–6.

Appendix. Algebraic Data Types

An algebraic data type (ADT) encapsulates the inductive definition of a data type in terms of concrete values, type parameters, product types ($\times$), and choice types ($+$). A few examples of ADTs are given here, with pointers to further research on the topic.

The list type,

$$List\ a = Nil + (a \times List\ a)\,, \tag{A-1}$$

is a fairly comprehensive construction, which uses all unarguably ADT features: parametricity, product, sum, and recursion. If $a', a'', ...$ are data of type a, then data of type $List\ a$ are of the form $Nil, a' \times Nil, a'' \times a' \times Nil$.... Special notation using square brackets and commas are often used for lists, so the previous sequence of lists is also written as $[\,]\,, [a']\,, [a'', a']$,....

Reading the type of $List\ \ a$, we see how it forms the rule for creating the example lists. The definition reads that a list with elements of type a are either a null list, Nil, or an element of type a and more list. The type parameter, a, on the left side of the definition lets lists be polymorphic in type. Examples are the list of integers, $[1, 3, 9] : List\, Integer$; and the string, "$word$", represented as lists of characters, $['w',' o',' r',' d'] : List\ Char$.

To programmers coming from the imperative world (e.g., C, Java, Python, Matlab, and Ruby), the most familiar element of constructing an ADT is undoubtedly the product, $\times$. It is used to create data with multiple fields as in C structures. For instance, a circle defined by a center point and radius can be represented with a triple of floating point numbers, $Circle = Double \times Double \times Double$. Of course, useful languages let programmers label the fields (accessors) so they make fewer mistakes programming with the data type.

The sum, +, (co-product) type allows a programmer to specify between two alternatives. This was seen already in the definitions of $List\ a$ and $info$. Using parametric polymorphism, the canonical choice between any two types can be given as $Either\ a\ b = a + b$. Another special class of co-products are enumerated types of the form, $T = 1 + 1$. As in the case of products, languages allow labeling of the choices for the sake of program consistency. For instance, $Bool = True + False$ is isomorphic to T.

ADTs get their name due to theoretical foundations in universal algebra, which describes algebraic structures as a triple consisting of a carrier set, X; a set of functions, Σ, from n-tuples of X-es to X; and a set of equations over the elements of X. Category theory provided a unifying framework for universal algebra, describing algebras as a carrier, X, and a co-product function, α, from a functor, F, to X: $\alpha : FX \to X$. Initial algebras are of special interest. They are algebras, $\alpha : FX \to X$, satisfying that for any other F-algebra, $\beta : FX \to X$, there is a unique mapping from $\alpha \mapsto \beta$.

They provide the basis for both specifying data via an ADT and proofs via induction for statements about ADT data.

The categorical treatment of data via initial algebras indicates a dual notion, called final co-algebras. Co-algebras have been shown useful for modeling transition systems, infinite streams, and context free languages. Generally, "co-data" models objects that we cannot construct but can observe and manipulate via an abstract interface. As such, it may be desirable to model some information such as $Location$ or $Time$ as co-data.

Co-algebras also provide a proof principle dual to that of induction, co-induction. While data is formalized using denotational semantics relying on initial fixed points, final co-algebras provide formalization of the operational semantics of co-data, relying on final fixed points. An introduction to initial (and final co-)algebra can be found in Jacobs and Rutten.[1] Rutten has expanded the theory of universal co-algebras[2] and many more example applications can be found in the publications on his web page at Centrum Wiskunde & Informatica.[3]

It is a stated intent of this report to at least hint how the use of type signatures is interwoven with some deep mathematical theories. We have already seen an application of category theory in the design of data. Category theory also provides further insights for the modeling and study of computation. Notable among these are the Monad,[4] which is a composable model of computational contexts such as failure conditions, input/output (IO), and mutable state; and the Functor, modeling abstract containers that can be mapped over, such as trees and lists.

There is a vein in the computer science research for generating functions based on data definitions. That is, building general recursion schemes based on folds (aka map reduce) and unfolds.[5] More recently, Hinze[6] has demonstrated generalized recursion schemes based on adjoint functors. Such efforts are aimed at providing program assurances such as termination and have lead to optimizations such as stream fusion.[7,8] Another interesting application for ADT is in the deriving a traversal

[1] Jacobs B, Rutten J. A tutorial on (co)algebras and (co)induction. EATCS Bulletin 1997;62:62–222.

[2] Rutten JJMM. Universal coalgebra: A theory of systems. Theor. Comput. Sci. 2000;249(1):3–80.

[3] homepages.cwi.nl/~janr.

[4] Wadler P. Monads for functional programming. In: Advanced Functional Programming; Springer; 1995; p. 24–52.

[5] Meijer E, Fokkinga M, Paterson R. Functional programming with bananas, lenses, envelopes and barbed wire. In: Functional Programming Languages and Computer Architecture; p. 124–144.

[6] Hinze R. Adjoint folds and unfolds. In: Mathematics of Program Construction; p. 195–228.

[7] Hinze R, Harper T, James DW. Theory and practice of fusion. In: Implementation and Application of Functional Languages; Springer; 2011; p. 19–37.

[8] Coutts D, Leshchinskiy R, Stewart D. Stream fusion: From lists to streams to nothing at all. In: Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming; ICFP '07, New York, NY, USA: ACM; 2007. p. 315–326.

structure for a data type based on the derivative of that type,[9] which is, in turn, generalized for any data type representable as the initial fixed point of a polynomial functor.[10]

In summary, ADTs provide an adequate basis for modeling many types of data. They are founded in well-established mathematical theories and amenable to formal reasoning. Furthermore, the dual notion may prove useful for modeling objects not easily captured as data, i.e., systems.

[9] McBride C. The derivative of a regular type is its type of one-hole contexts.

[10] McBride C. Clowns to the left of me, jokers to the right (pearl): Dissecting data structures. In: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages; POPL '08, New York, NY, USA: ACM; 2008. p. 287–295.

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) US ARMY RESEARCH LAB ATTN IMAL HRA MAIL & RECORDS MGMT ATTN RDRL CIO LL TECHL LIB

1 (PDF) GOVT PRINTG OFC ATTN A MALHOTRA

1 (PDF) US ARMY RESEARCH LAB RDRL-CIN-T TREVOR COOK

INTENTIONALLY LEFT BLANK.