Much Ado About Nothing

A brief survey of the concept of nothingness from antiquity to modern systems

Kenneth Baclawski
Washington Academy of Sciences

Abstract

Whether it is called "nothing," "no value," "vacuum," "empty" or "zero" the concept of *nothing* has been a matter of philosophical as well as pragmatic disagreement since antiquity and continuing today.

Introduction

THE CONCEPT OF NOTHING, whether the abstract notion of nonexistence or the physical vacuum, has been debated since at least the fifth century BCE. The related notion of *zero* has an even longer history. One might think that such issues would have been settled long ago, yet in many ways the opposite is the case. The properties of the physical notion of a vacuum is currently a major unsolved problem. While the mathematical notions of zero and the empty set are well-established, modern computer programming and data languages differ markedly in how they deal with nothingness. The differences have a significant effect on how such languages behave when encountering nothingness, which is a continuing source of confusion. In this survey we outline some of the history of nothingness. We then discuss the main notions of nothingness in modern data and programming languages. Finally, we list some of the many explanations for why one might have nothing.

History

There are two distinct uses of zero. The first is for a place-holder in positional numbering systems when a place has no value. The second is for the absence of a quantity. The Sumerians had a base 60 positional numbering system as early as 2500 BCE, and used a space for a position with no value, but did not have a symbol for zero. By 1770 BCE, the Egyptians had a symbol for zero in accounting texts (Gheverghese, 2011), but their numbering system was not positional (O'Connor and Robertson, 2000). The Babylonians inherited the Sumerian positional numbering sys-

tem and subsequently introduced a symbol for a position with no value, but never used it alone. The ancient Greeks had neither a symbol for zero nor a positional numbering system, in spite of being aware of Babylonian mathematics (Wallin, 2002). The reluctance of the ancient Greeks to use zero may be due to their philosophical arguments about the nature of existence and the vacuum. This reluctance continued well into subsequent civilizations, including Medieval Europe. It wasn't until around 650 CE that Brahmagupta in India formalized arithmetic operations using zero (Wallin, 2002).

One of the earliest Greek philosophers to discuss the concept of nothingness was Parmenides at around 500 BCE. At this time nothingness meant nonexistence and included the physical void or vacuum. Parmenides argued that the very act of speaking about nothing implies that it exists, which contradicts the definition of nothing as nonexistence (Russell, 1995). On the other hand, as was pointed out by Leucippus, the co-founder of atomism with his student Democritus, one can only have motion if there is a void through which matter moves. It then follows that there must be a void between the atoms. Little is known about Leucippus, and there is a considerable controversy about whether Leucippus ever existed, starting with Epicurus at around 400 BCE and continuing to modern times (Skordoulis & Koutalis, 2013, p. 467; Graham, 2008, pp. 333–335). If there was no such philosopher as Leucippus, we have the ironic situation that the first person to speculate about the existence of nonexistence did not exist.

Unfortunately, the early Greek philosophers did not possess a technology that was adequate for experimentally testing the speculations of the atomists so the theory was not generally accepted at the time. Hero of Alexandria tried unsuccessfully to create an artificial vacuum in the first century CE, and partial vacuums were not demonstrated until the 17^{th} century. Extremely high artificial vacuums can now be achieved, and outer space is an even more extreme vacuum than artificial vacuums, but even if all particles of matter were removed from a small volume, there would still be photons and neutrinos, as well as dark energy, virtual particles, and other aspects of the quantum vacuum. While a perfect vacuum is not currently regarded as being physically realizable, it is meaningful to speak of the lowest possible energy state of physical space, which is called the quantum vacuum, zero-point field or vacuum state. The properties of

the vacuum state is currently a major unsolved problem which has been very well covered by Sethanne Howard in another article in this same issue (Howard, 2023).

As with the physical notion of a vacuum, the ontological notion of nonexistence is also an unsolved problem, or at least a problem that remains controversial. After the Greeks, many philosophers attempted to understand nothingness, sometimes giving the notion a religious or spiritual interpretation. John the Scot (c. 815–877) identified nothingness with God (Russell, 1995, pp. 396-401). Hegel in his *Science of Logic* asserted that "pure nothing" is the same as "pure being," but he then contradicted this assertion in the next section by asserting, "But it is equally true that they [pure nothing and pure being] are not undistinguished from each other, that, on the contrary, they are not the same…" Hegel then proceeds to resolve the contradiction by introducing the relationship of "becoming" between pure nothing and pure being. To Hegel pure being is closely related to Geist, which can be roughly translated into three English meanings: ghost, spirit, and mind or intellect. (Hegel, 1812-1816, §§ 132-134).

Modern philosophers have continued to grapple with nothingness. In 1929 Martin Heidegger gave his Freiburg inaugural lecture entitled "What is Metaphysics?" (Heidegger, 1929). In this lecture he announced "Das Nichts selbst nichtet" (Cartlidge, n.d.). Understanding this or translating it to English is problematic because Heidegger commonly invented new words to help convey his ideas. Unfortunately, doing this has made it difficult to understand his intentions not only in his native language but also in translations to other languages. The problematic word in Heidegger's announcement is "nichtet," which can be interpreted as the verb form of the word "nicht" ("not" in English). As there is no such verb in English, it has been variously translated as "annihilates," "nothings" or "noths," where the last one is a back-formation obtained from the word "nothing" when regarded as being the gerund of a verb. This leads to the speculation that Heidegger was attempting subtle humor by asserted the existence of nonexistence using a nonexistant word, but that seems unlikely. In any case, one can translate the announcement variously as "Nothingness itself annihilates," "Nothingness itself nothings" or "Nothingness itself noths."

Here are a few quotes from Heidegger's lecture as translated by

Goth (2013). To improve readability, hyphenated terms in the translation were changed to common English words, e.g., "no-thing" was changed to "nothingness."

> But why do we trouble ourselves about this nothingness? In fact, nothingness is indeed turned away by science and given up [on] as the null and void [das Nichtige]. But if we give up nothingness in such a way, do we not indeed accept it? But can we talk about an acceptance if we accept nothing [nichts]? Yet maybe all this back and forth has already turned into empty verbal wrangling. Science must then renew its seriousness and assert its soberness in opposition to this, so that it has only to do with being [um das Seiende geht].

Note how Heidegger appears to be harkening back to the argument of Parmenides. Heidegger then repeats the argument of Parmenides in various ways, as for example the following:

> Accordingly, every answer to this question is impossible from the outset. For it necessarily starts out in the form: nothingness "is" this or that. Question and answer alike are themselves just as nonsensical with respect to nothingness.

Finally, he announces "Das Nichts selbst nichtet" but makes it clear that the annihilation is not an occurrence of some sort whereby something else annihilates or negates. Nothingness does the annihilation on its own.

This soon earned Heidegger fame as a purveyor of metaphysical nonsense (Inwood, 1999). Nevertheless, Heidegger is often considered to be among the most important and influential philosophers of the 20^{th} century. One interpretation of Heidegger's announcement is that in spite of the self-contradictory aspects of nothingness as a concept, one can nevertheless regard it as have its own kind of existence.[1] In any case, I am reluctant to claim that Heidegger's announcement is not unmeaningless, since another philosopher subsequently developed this idea into a rich body of popular work.

[1] I suspect that Heidegger would disagree with this interpretation.

Jean-Paul Sartre was one of the many philosophers who were influenced by Heidegger. Sartre is regarded as being the most prominent existential philosopher, and his principal philosophical work is *Being and Nothingness* (Sartre, 1943), an obvious reference to Heidegger's principal work *Being and Time* (Heidegger, 1927). As the title suggests, nothingness is central to Sartre's work. Beginning in the first chapter, Sartre points out that nothingness is something we experience as part of reality. The absence of a family member or a lack of money are not just subjective nothings. Starting from this thesis, Sartre developed a wide range of topics such as consciousness, perception, social philosophy, self-deception, psychoanalysis, and the question of free will. Sartre was a prolific writer having written novels, short stories, plays, screenplays, autobiographical works as well as philosophical and other essays. His work is richly symbolic and clearly shows its basis in his philosophy.

There have been many other philosophers who have discussed nothingness, but the examples in this section should give one an inkling of the kinds of arguments that have been presented. None of the arguments have been entirely refuted. Indeed, the argument of Parmenides can be used to assert that a category is meaningful only if it potentially could have members. For example, one could argue that the category of unicorns is not meaningful, although the category of articles about unicorns is meaningful. On the other hand, one could allow for the existence of nothingness as asserted by Hegel, Heidegger and Sartre. We now give some examples of both approaches in modern systems.

Nothingness in Information Systems

Modern computer languages differ in how they deal with nothingness. The main distinction is whether nothingness should be regarded as a special kind of value (as asserted by Heidegger and Sartre) or should be no value at all (as suggested by Parmenides). The choice between these two is neither trivial nor obvious, and there are important advantages and disadvantages for the two choices. Specifically, most major programming languages have chosen to treat nothingness as a special value, while data languages, such as the standard relational database language SQL, have generally chosen to treat nothingness as the absence of a value.

Programming Languages

Historically, a notion of nothingness in programming languages was needed when programmers began to symbolically specify addresses (locations) of data in memory. There needed to be some default address when the symbol does not specify any data, for example, because the data has not yet been allocated a location in memory. Originally, this default address was the address 0. The C language still allows one to specify the default address with the digit 0 even though the actual default machine address need not be an address all of whose bits are 0. Other programming languages use names such as "null," "NULL" or "nil." Even in C it is now considered preferable to use the NULL symbol rather than the digit 0. Regardless of the specific name used, they are called null pointers or null references. Tony Hoare invented the null reference in 1965 as part of the ALGOL W language (Hoare, 2009).

Null pointers are commonly used to represent conditions such as the end of a sequence of actions having unknown length or the result of an action that did not produce any data. An important requirement for nulls is that one can test for them by a simple comparison. In particular, if two program variables that both have the null value are compared, then the result is TRUE.

Accessing the data referenced by a data pointer is known as "dereferencing." Because a null pointer does not point to a meaningful object, attempting to dereference it may cause a run-time error or program crash. This is called a null pointer exception. It is one of the most common types of software weakness. Because of this Tony Hoare referred to his invention of the null reference as a "billion dollar mistake" (Hoare, 2009).

While dereferencing a null reference is an exception in most programming languages, this is not always the case. For example, in Lisp the default value is called nil, and it can be used like any other value with well defined results and no exceptions. Python is similar to Lisp except that nil is called "None." However, in either of these languages the disadvantage is that if a use of nil/None was actually an error, then one could get into an infinite loop, which is arguably worse than a program crash. Objective-C is another language where dereferencing nil is not an exception, although nil is not treated as being an ordinary value. Unlike Lisp, dereferencing nil

is ignored in Objective-C. As with Lisp, erroneous uses of nil could result in an infinite loop.

For most programming languages where deferencing a null pointer causes an exception, one can catch the exception and take appropriate corrective action. Arguably it is better to generate an exception for an invalid dereference operation than to process it or to ignore it, which will just postpone the problem and make finding the error much more difficult.

Relational Database Languages

The notion of nothingness that is part of modern relational databases was introduced by Codd's original proposal in 1970 for the relational model (Codd, 1970), which is nearly contemporaneous with Hoare's invention of the null reference. The relational symbol for the lack of a value in the field of a record is NULL.

Ever since Codd introduced NULL, it was controversial. The main problem is that the concept is underspecified, resulting in inconsistent implementations. Even to the extent that it is specified, it is inconsistent, and explanations for the inconsistencies are not convincing. Many researchers have published, and continue to publish, proposals for correcting the difficulties with NULL. For example, Date and Darwen advocated in their Third Manifesto that the concept be eliminated entirely from relational database systems, replacing it with the programming language concept (Darwen and Date, 2006).

So what is the database NULL? As we mentioned in the previous section above, a database NULL is the absence of a value, while for most programming languages, null is one more value, albeit a special one. This has a number of consequences. The first is that comparing a database NULL with another database NULL, even the same database NULL, results in NULL, not TRUE. This makes sense because a database NULL means that there is no value, so comparisons are meaningless. One consequence is that one cannot use ordinary Boolean logic. The result of a logical expression now has three possibilities rather than just two. In addition to TRUE and FALSE, one must now admit the possibility of NULL. The resulting logic is an example of a three-valued logic, while ordinary Boolean logic is a two-valued logic. Strictly speaking the third "value" of the SQL

three-valued logic is not a value but rather the absence of a value, but for the purposes of evaluating a logical expression it is convenient to regard the third possibility as being another value.

One of the consequences of the three-valued logic is that one cannot test that the field of a record has a value by comparing the field with NULL. Doing so will always produce NULL whether the field has a value or not. Relational database systems have a special syntax for this purpose. One must say "field IS NULL" or "field IS NOT NULL."

A more significant problem with the database NULL is that it has led to confusion. For example, the term "null value" is an oxymoron for databases. Yet the phrase is used throughout database textbooks and articles on the Web such as in Wikipedia. Textbook writers even go so far as to make statements such as, "Nulls behave strangely in queries." The strangeness is entirely due to mistakenly believing that the database NULL is a value. When one properly understands that the database NULL is the absence of a value, then the strangeness disappears. Another example is, "A weird aspect of nulls is that they are neither equal nor unequal to each other." Once again, the weirdness is due to a misunderstanding rather than an intrinsic property of databases.

Unfortunately, while SQL generally regards NULL as being the absence of a value, it is not entirely consistent. For example, when records are sorted, NULLs in different records are regarded as being equal.

Even Codd himself admitted that his concept of NULL was flawed, and he attempted to fix it by introducing two different concepts of NULL (Codd, 1990). However, this suggestion was never implemented, largely because of the enormous increase in complexity that it would entail. For example, logic would have to increase from being three-valued to being four-valued. Furthermore, Codd's suggestion would not have fixed the various inconsistencies involving NULL. As we discuss below, it has the further disadvantage that there are many more than just two NULL concepts, and even the two NULL concepts suggested by Codd have their own subclassifications. Attempting to fix the problem by making more distinctions is clearly not a solution.

It is interesting to note that the inventors of their respective notions

of nothingness both regarded their inventions as a mistake.

Other Data Languages

More recently, data representations and data languages other than the relational databases have become popular. Prominent examples include the RDF and OWL representation languages as well as query languages such as SPARQL. For the rest of this section, we assume some familiarity with RDF and OWL.

A fundamental distinction between relational databases and RDF/OWL is whether they are open or closed. A logical system satisfies a closed world assumption (CWA) if some statements that are not currently known to be true can be inferred to be false. By contrast if a logical system satisfies the open world assumption (OWA), then lack of knowledge need not imply falsity. There are many kinds of CWA logical systems as well as many kinds of OWA logical systems, as well as logical systems that have aspects of both assumptions. The advantage of CWA is that query processing is much more efficient compared with OWA. However, CWA requires that a situation be completely known to the database. This is acceptable for a business database but is not reasonable for general knowledge queries.

A relational database is primarily CWA, but NULL can be regarded as an OWA feature in the midst of a system that is otherwise CWA. Most programming languages may be regarded as being CWA. Unlike relational databases, programming language nulls are values and so cannot be regarded as being an OWA feature. RDF, OWL and their query languages are defined to be OWA logical systems, although some aspects of CWA are sometimes employed for the sake of efficiency.

To give some idea of the consequences of OWA compared with CWA, consider a university database where students may have a major in a department, but the university has decided that no student can major in more than one department. Suppose that in spite of this rule a student has declared that they are majoring in English and Computer Science. In a CWA system, this would immediately result in an error because the major constraint is not satisfied. In an OWA system, this might not be an error. Instead, the OWA system would proceed to infer that English and Computer Science are the same department. A cascade of other inferences

might then ensue. For example, if a department is allowed to have no more than one head, then the heads of English and Computer Science would be inferred to be the same. Presumably, the cascade of inferences would eventually result in an unacceptable inconsistency, but by then the original reason for it might not be easy to trace.

In RDF and OWL the most common way to specify that a resource does not have any value for a property is to have no statement that has the resource as its subject and the property as its predicate. This is essentially the same as a database record having a NULL field. Indeed, some relational database implementations represent the fact that a field is NULL by omitting the field in the record.

Another way that one can, in principle, specify that a resource does not have a value for a property is to specify that its value is an empty list. The advantage of the list construct is that it allows one to specify exactly all of the values in a collection. In particular, it could be used to specify exactly all of the values that a resource has for a property. Accordingly, a list can be regarded as a CWA feature in the midst of a system that is otherwise OWA.

Some Interpretations of Nothingness

Whether in programming languages or data languages, NULL can arise for a great variety of reasons, and attempting to capture even a few of these in a language would be very difficult if not impossible. Here is a partial list of the reasons why NULL can occur.

1. The field has a value but it isn't known. There are a number of variations on this. Here are a few of them:
   (a) The value does exist somewhere, but the database does not have it.
   (b) The value has not yet been determined, but it presumably will be determined at some point. For example, a customer has yet to make a decision about a purchase.
   (c) The value does exist in principle, and it might someday be determined, but there is no guarantee that it will ever be determined.

For example, a field might have value 1 if P=NP and 0 if P≠NP.[2]

All of these may be regarded as examples of open world assumptions in the midst of a system that is otherwise closed. They have the common feature that the field could, in principle, be specified with a value at some time in the future.

2. The field has no value because it isn't applicable. Unlike the example above, there is no possibility that such a field could ever have a value. This can happen for a variety of reasons.

   (a) The field is not allowed to have a value. For example, the root node of a tree has no parent node. The fact that the field is NULL in this case is not due to missing information. As another example, in programming languages, a null reference is sometimes employed to mark the end of a sequence of objects.

   (b) The field was produced during an operation such as an arithmetic or data manipulation operation.

3. The field has a value, but it is not within the domain. For example, a form requesting an ethnic group does not include the one to which a person most closely identifies. The NULL in this case represents "none of the above" or "other." On such a form, NULL could also mean that one does not wish to answer the question, one does not know the answer, one neglected to answer the question, or many other possible explanations.

4. A field value could have been inferred but was not inferred, because of an overriding requirement. The example given here requires some knowledge of SQL. Specifically, it requires the outer join operation. The outer join mandates that the additional (padded) columns of an unmatched record be NULL even when one could infer values for the columns. Consider the following SQL statements:

[2]The P=NP problem is an unsolved problem in computer science. It asks whether every problem whose solution can be quickly verified can also be quickly solved, where "quickly" means "achieved in a period of time that is a polynomial in the size of the problem."

```
create table A(id int primary key, foo int default 0);
create table B(id int primary key, bar int default 0);
insert into A values(1, 10);
select * from A left join B on (A.id = B.id);
```

The result of the SQL statements above is the following:

id	foo	id	bar
1	10	NULL	NULL

(a) The second id field in the result must be NULL because the join condition did not hold in this case. It means that the A record was not matched with any B record. This represents a situation where a field cannot have a value.

(b) The bar field in the result could be inferred to be 0 because of the default value for this column. In spite of this, the bar field in the result is NULL. One could argue that this is also a situation where a field cannot have a value. In this case, there is no B record being joined, so it is not meaningful for it to have any value, even the default value. On the other hand, the usual purpose of a default value is that it is the value one should use in the absence of any other value being available. That certainly is true in this case, so there is also a good argument for using the default value rather than NULL.

5. The field value cannot be determined due to an exception. For example, dividing 1 by 0 is clearly an exception since division by zero is never defined for any number.

There are many other possibilities for the interpretation of nothingness. During query processing, computations use a three-valued logic which will introduce NULLs that have interpretations that are subtly different from the interpretations of the NULLs that occurred in the expressions being computed. For example, if NULL means that the value has not yet been specified, then the result of join processing should be that there could be a match but it is not yet known. By contrast, if NULL means "none

of the above," then one could argue that two NULLs should match one another.

One reason why the database notion of NULL is still useful in spite of its semantic shortcomings has to do with query optimization. Unlike programming languages which have relatively limited opportunities for optimization, there are substantial opportunities for optimizing queries. For example, a query that filters the results as the last step could be optimized so that the filter operation is performed much earlier in query processing, well before operations such as joins are performed. While this has important consequences for performance, it also means that standard programming language features like throwing and catching exceptions are no longer possible. By rearranging the processing steps the way databases do, one could be in a situation where the exception is caught at a different place in the computation than one would expect. In other words, the throw-catch notion is only meaningful for a specific execution plan, so throwing and catching exceptions within the processing of a query would mandate a specific execution plan and eliminate many possibilities for query optimization.

Still another kind of optimization is parallel processing (via multiple threads) to obtain query results. When an exception is thrown in a parallel context, it is unclear how the exception should be handled. Aside from the issue of where the exception should be caught, there is also the issue of whether other threads should be affected by the exception. The use of NULL and NULL propagation rules allows query processing to be optimized and parallelized, but at the cost of a notion of NULL that differs from the one for programming languages as well as a logic that is three-valued rather than two-valued.

Conclusion

The notion of nothingness has a long and complex history. It is both a philosophical issue that continues to be argued and a pragmatic issue that continues to have an impact on modern systems. We have given some idea of the difficulties faced by anyone attempting to specify the semantics of nothingness in a way that satisfies all relevant interpretations and use cases, while being compatible with the requirements for system performance.

References

Cartlidge, J. (n.d.), A summary of Heidegger's "What is Metaphysics?" https://bit.ly/3FwAPzA

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* **13**(6): 377-387.

Codd, E.F. (1990), *The Relational Model for Database Management Version 2*, Addison-Wesley, 1990 (ISBN 0201141922).

Darwen, H. and Date, C.J. (2006), *Databases, Types, and The Relational Model: The Third Manifesto*, 3^{rd} edition, Addison-Wesley. (ISBN: 0-321-39942-0).

Gheverghese, J. (2011). *The Crest of the Peacock: Non-European Roots of Mathematics (Third ed.)*. Princeton UP. p. 86. ISBN 978-0-691-13526-7.

Goth, M. (2013). Translation of Heidegger's "What is Metaphysics?" and other essays. https://bit.ly/3MeNB9d

Graham, D.W. (2008). "Leucippus' Atomism". In Curd, Patricia; Graham, Daniel W. (eds.). *The Oxford Handbook of Presocratic Philosophy.* Oxford University Press. ISBN 978-0-19-514687-5.

Hegel, G.W.F. (1812-1816), *Wissenschaft der Logik*. https://bit.ly/47t41mt

Heidegger, M. (1927), *Sein und Zeit (Being and Time)*.

Heidegger, M. (1929) "What is Metaphysics?"

Hoare, T. (2009). "Null References: The Billion Dollar Mistake." InfoQ.com.

IEEE 754 (2019). IEEE Standard for Floating-Point Arithmetic. IEEE STD 754-2019. IEEE. pp. 1–84. IEEE Computer Society (2019-07-22). https://bit.ly/47ckjAK

Inwood, M. (1999). Does the Nothing Noth? Royal Institute of Philosophy Supplements, **44**: 271-290. http://bit.ly/3yLv6SS

Menninger, K. (1992). *Number words and number symbols: a cultural history of numbers.* Courier Dover Publications. pp. 399–404. ISBN 978-0-486-27096-8.

O'Connor, J. and Robertson, E. (2000). "Egyptian numerals". mathshistory.st-andrews.ac.uk. https://bit.ly/45N3JWz

Russell, B. (1995). *History of Western Philosophy*, Routledge ISBN 0-415-07854-7.

Sartre, J.-P. (1943), *L'Être et le Néant (Being and Nothingness)*. Trans. Hazel E. Barnes. New York: Washington Square Press, 1984.

Skordoulis, C.D., Koutalis, V. (2013). Tsaparlis, Georgios (ed.). *Concepts of Matter in Science Education.* Springer. ISBN 978-94-007-5914-5.

Wallin, N.-B. (2002). "The History of Zero." The Whitney and Betty Macmillan Center for International and Area Studies at Yale. https://bit.ly/40kL9UM