

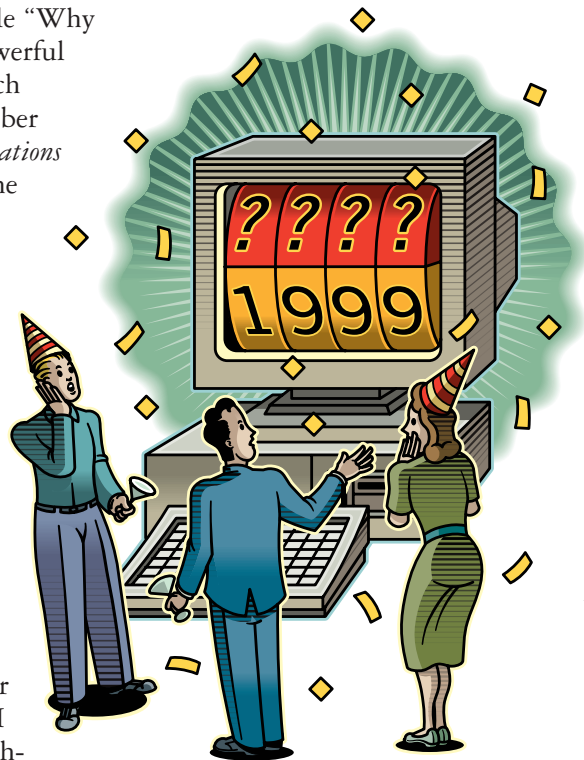
Hal Berghel

# The Year-2000 Problem and the New Riddle of Induction

Normally, my columns are a product of accretion. This one, however, was interrupt driven. The trigger in this case was the convergence of two independent events: my response to Peter Wegner's article "Why Interaction Is More Powerful than Algorithms," which appeared in the September 1997 issue of *Communications* ("Forum," p. 20) and the publication of an ACM Press book by Capers Jones, *The Year 2000 Software Problem: Quantifying the Costs of Assessing the Consequences*. Not one to ignore kismet, I wrote this column.

The story unfolds this way. Following the publication of his article, Wegner and I were discussing various computational metaphors for interactive computing. I found Wegner's "algorithmic computing is weaker than interactive computing" argument compelling. In addition to his formal arguments showing that interaction machines cannot be expressed by Turing machines and his incompleteness proof that

interactive systems cannot be expressed by first-order logic, there are several practical examples showing that interactive services like banking or airline



reservations cannot inherently be realized by non-interactive (Turing machine) systems.

I suggested likely examples that went beyond the inadequacy of algorithmic computing in

handling the interactivity. One that came to mind was the year-2000 (Y2K) problem. I suggested that computer users (both physical and digital) take each system report of a correct date as a confirming instance of the hypothesis that the system's reported date is always correct.

What the Y2K problem shows is that correct time observations were not confirming instances of this hypothesis at all, but rather one where "date" terms were unknowingly temporally qualified.

That is, we assume system date reports state, "this is the current date," when in fact the report should have been interpreted as "this is the current date only if the current date is prior to year 2000." Most experimental computing is predicated on the fact that successful runs confirm correctness, and I was trying to show the Y2K problem provided a counterexample. My promised elaboration was triggered by the appearance of the Capers Jones book.

## The Architecture of the Y2K Problem: A Few Bytes—No More, No Less

What's the true storage cost of

**THE Y2K PROBLEM IS ABOUT A COUPLE OF BYTES OF STORAGE saved in databases, a couple of bytes saved on silicon, a couple of bytes saved in BIOS routines, a couple of bytes spared by operating system function calls, a couple of bytes of I/O. This problem is about a couple of bytes.**

two bytes? Minuscule in the individual case, wasteful excess that multiplied by the billions. The Y2K problem is about a couple of bytes—a couple of bytes of storage saved in databases, a couple of bytes saved on silicon, a couple of bytes saved in BIOS routines, a couple of bytes spared by operating system function calls, a couple of bytes of I/O. This problem is about a couple of bytes.

The result of the computing community's byte parsimony is a multitude of computing systems that will not, in the normal course of things, roll over to 2000 at the end of this century. Many systems will roll back to 1900 instead. Others might pass beyond the millennium threshold correctly, only to fail to roll over from 2099 to 2100. Still others may fail to advance beyond a predetermined elapsed time in seconds since a certain starting date. These anomalies are motivating organizations worldwide to prematurely retire their computer systems, scramble for patches and work-arounds, and establish Y2K rapid response teams to deal with all of the glitches the other techniques fail to address. The aggregate worldwide cost of dealing with this problem is staggering—by some estimates in the hundreds of billions of dollars—but more on that later. How did we come to this?

Flashback to the 18th-century Scottish philosopher, David Hume. (This is going to be a bit of a stretch, but bear with me. In a few pages we will converge on a point of enlightenment, or at least raise the Y2K problem to a lofty theoretical level it may not deserve.)

Hume was skeptical about finding a foundation for inductive reasoning—the inference from particular to general. We can't come to "know" that the sun will rise daily in the same way as we "know" the equality of  $2+2$  and  $4$ , he argued, because reason is not the source of inductive beliefs: induction is neither rational or logical. Hume wondered, how could one justify induction. Clearly it was beyond the capacity of deductive reasoning. But an inductive justification would be viciously circular. Thus he concluded that inductive reasoning can't be justifiable in the customary (that is, deductive) sense of the term at all. So we have, as it were, an inductive paradox that, for convenience, we'll call the "old riddle of induction." Inductive reasoning works, Hume thought, not because it involves the justified inference from the particular to the general, but because the cause-effect relations and associations involved are internalized in some sort of "instinct." The legitimate expectations on which our behav-

ior is based—that the sun will rise in the East tomorrow at about the same time as today, that the trade winds will continue to blow, that the seasons will change—are all expectations founded on this human instinct.

What does Hume have to do with the Y2K problem? Be patient and read on.

### **Inductive Riddles**

So, Hume leaves us not only with his "old riddle of induction," but also with an escape clause: we can account for our faith in inductive reasoning by appeal to basic instinct. Induction isn't rational, but it works so well as a foundation of our beliefs that we needn't worry much. The "old riddle of induction" dissolves. Is that the end of the matter? In a word, no. A paradox remains which will tie in to the Y2K problem.

Flash forward to the mid-1900s, when a Harvard philosopher, Nelson Goodman, investigates Hume's work and concludes that Hume has conceded too much to his critics. The "vicious circle" isn't really vicious at all. Goodman claims that inductive reasoning, just like deductive reasoning, is justified by conformity to appropriate, general rules. If, for example, we predict tides that don't arrive, we adjust the rules that gave rise to the prediction. So it is with

deduction. If the axiom of specification produces a contradiction (a.k.a. Russell's Paradox), we substitute another less-problematic axiom, or introduce a set theory based on types, or whatever else is necessary to fix the problem. Induction should fare no worse than induction in terms of its justification.

However, just as the old riddle is solved (or dissolved), Goodman finds a new riddle. The new riddle has to do with hypothesis testing.

In inductive reasoning, hypotheses are both generalizations of, and predictors of, evidence statements. As Goodman observes, the fact that one copper wire conducts electricity is confirming of the hypothesis that copper conducts electricity, but the fact that one student in class is a third son does not confirm the generalization that all students in the class are third sons. Generalizations from evidence statements are only possible when the hypotheses are law-like.

Suppose, Goodman suggests, that we observe that all emeralds before some time,  $t$ , are green. Each observation of a green emerald prior to  $t$  is therefore a confirming instance of the hypothesis that all emeralds are green. Next, let's introduce another predicate, "grue," such that an object is grue just in case it is green before  $t$ , or blue thereafter. The same observations now support the claim that all emeralds are grue. Both predictions are equally supported, but those involving "grue" are not supported to the same degree. The reason is that the instances of grue-ness betray a linguistic coin-

idence, rather than a law-like regularity.

The new riddle of inductive reasoning (a.k.a. Goodman's Paradox) derives from the fact that we cannot always distinguish law-like regularities from contingent or accidental ones. We have no straightforward way to distinguish between the case where light bending around the moon during a solar eclipse confirms Einstein's General Theory of Relativity, than Aries' presence in the house of Mars with solar exaltation confirms a cookie's fortune.

## The Y2K Problem Revisited

You've stuck it out this far. It's downhill from here.

The genesis of the Y2K problem is seen to be an instance of the "new riddle of induction." It results from the mistaken belief that a computer date stamp is projectable—that is the confirmation of a hypothesis. We blithely assumed that an operating system date stamp satisfied the predicate "\_\_\_ is the current date on the Gregorian calendar" when, in fact, the predicate was "\_\_\_ is the current date on the Gregorian calendar only before time,  $t$ ; and not thereafter." The accuracy was contingent and not law-like.

Actually, the problem is even worse than that. Sometimes there are several layers of non-projectable predicates involved. I'll use DOS to illustrate the point. We take the result of function 2Ah of interrupt 21h ("Get Date") to be a confirming instance of the hypothesis that this function produces the current date in register CX. Such is

not the case. Unbeknownst to us, DOS only recognizes the contents of CX if it falls within the range 1980–2099 (2099 is time  $t$  in this case). But the reason for this constraint betrays even more convoluted logic beneath DOS. Function 2Ah retrieves date information from the ROM BIOS services. However, subservice 04h of BIOS interrupt 15h ("Get Real-Time Clock Date") actually only returns the century (either 19 or 20) in register CH and the half-word integer equivalent of a two-digit year in CL. So the 4-digit integer date that the DOS function reports is already the product of interpretation by the OS. But it doesn't stop there. The DOS FILE\_DATE values, offset 18h from in the directory entry, is an even more corrupted version of a date stamp. In this case, the reported date is a compression of the DOS reported date according to the formula  $((\text{year}-1980) \times 512) + (\text{month} \times 32) + \text{day}$ . Hence, December 31, 1999 becomes the unsigned word integer 279Fh (10,143). It is interesting to note that the rollover for the file date would actually be  $1980 + 2^{**7}$  years, or 2108, but since DOS will only recognize the 19th and 20th centuries, the file date will be undefined under DOS beyond 2099. The point is that none of the associated predicates of these functions and interrupts are projectable beyond some time,  $t$ , in the near future. The new riddle of induction rears its ugly head in countless ways. The new riddle is to be found in the very bowels of our BIOS.

These sorts of problems exist in all complex interactive environ-

# Digital Village

ments. Some, like those I've mentioned, may be a result of incorrect software interpretations of hardware. Most, however, will be a result of semantic confusions engendered within software systems such as non-monotonic code expansion and confusing extensional and intensional meanings of variables.

## Economics of Byte Conservation

The root cause of our present malady is byte conservation. The lengths to which we have gone to save a few bytes would do justice to endangered species. But we may not have accomplished much in the end. Capers Jones quotes Leon Kappelman who has calculated that the cost to fix the Y2K problem will eat up all of the savings accumulated by compressing dates in the first place (see *Communications*, Feb. 1998, p. 30). Jones adds, "The Year-2000 problem actually originated as an explicit requirement by clients of custom software applications and the executives responsible for data centers as a proved and seemingly effective way of saving money. Many programmers knew that the clock would run out..."

So what are the estimated costs? Capers Jones offers a wealth of information on this subject. Consider his prediction of U.S. repair costs for the Y2K problem in Table 1.

Jones predicts a \$70 billion loss in the U.S. alone for Y2K-related software problems. What's more, this ignores the costs of repairing databases and data warehouses—costs he hypothe-

the moment that all occurrences of the Y2K problem are solved by the end of 1999. Will the problem go away?

Not necessarily. The real problem is that computer practitioners confuse the computer system's "interpretation" of a system predicate (`_IS_CURRENT_DATE`) with the actual predicate (`_IS_CURRENT_DATE PRIOR TO T; UNDETERMINED THEREAFTER`). Even if the Y2K problem disappears, we will still have to deal with the "elapsed time in Unix" problem. A manual might say that Unix function  $x$  produces today's date in Register 1. In fact, it won't "report" the date at all, rather it will "calculate the date on the basis of the number of seconds which have expired

since January 1, 1970, until count,  $t$ , reaches  $2^{*}31$ ; and the number of seconds since January 18, 2038 thereafter until count,  $t$ , reaches . . . etc." Naively, we have designed our programs for the projectable predicates described in our manuals, and not for the non-projectable predicates undermining them.

To make matters more complicated, there are several four-byte `REPORT_DATE` standards (ISO, Microsoft, European) in use that are all incompatible with one another. Will manuals of the future report that function  $x$  reports "current date," on the one hand, or "current date only if one

Table 1. U.S. repair costs for the Y2K problem

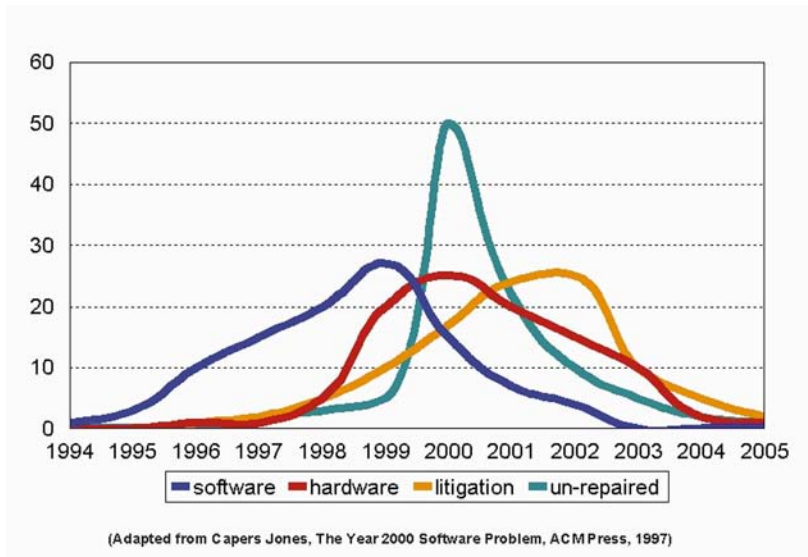
Industry	Effort (In person months)	Costs (In \$billions)
Military	1,909,091	\$14.3
Finance	450,000	4.9
Manufacturing	555,556	4.7
Communicatoins	423,529	4.2
Services	555,556	4.4
Insurance	450,000	4.1
Wholesale	517,647	3.9
Federal	400,000	3.2
Defense	266,667	2.9
Retail	412,500	3.1
Software	193,421	1.7
Municipal	150,000	1.1
Health care	111,563	.89
States	100,000	.77
Energy	87,500	.70
Transportation	82,031	.66
Other	1,800,000	15.1
<b>TOTALS</b>	<b>8,465,060</b>	<b>\$70,753,562,795</b>

(Adapted from Capers Jones, *The Year 2000 Software Problem*, ACM Press 1997. Used with permission.)

sizes could reach another \$125 billion. Figure 1 shows how these expenses will break out by year by type. As can be seen, the litigation costs of the Y2K problem only begin to express themselves. Figure 2 shows how these expenses break out by programming language as a percentage of the \$70 billion costs.

## The Consequences of the Y2K Problem

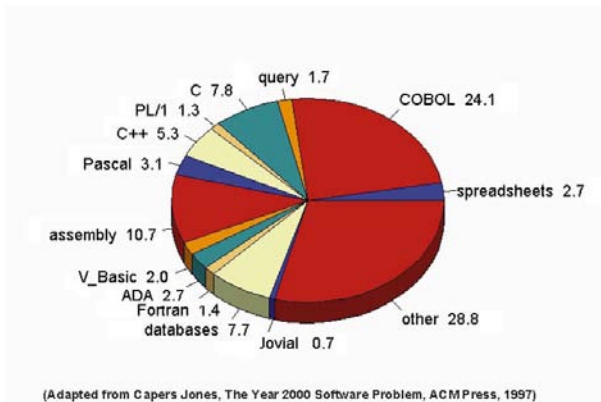
We illustrated the Y2K problem by means of a general problem with inductive reasoning called the "new riddle of induction." We also saw how this problem will affect our lives. Let's assume for



**Figure 1.** Y2K expenses by year by type

is using version,  $b$ , of operating system, OS $k$ , prior to time,  $t$ , in Microsoft format,” on the other?

In retrospect, a viable solution to the Y2K problems (in general) was to have been found in geochronology all along. The earth’s geologic time clock started roughly four billion years ago. Assuming that we are not yet at the mid-point of the earth’s evolution we need to keep track of less than  $10^{18}$  seconds or  $3 \times 10^{11}$  years which, for all practical purposes, can fit within 64-bits. Had system designers and architects been inspired by geochronologists, they would have selected a 64-bit date field to begin with. Even this might have been excessive. The dinosaurs, for example, only lived for 200 million years and they demonstrated far less propensity for self-destruction than humans. If we don’t outlive them, we could save four bits! In any case, according to Jones’s estimates, if



**Figure 2.** Y2K expenses by programming language (as percentage of total cost)

the use of this double-word, expanded date field gobbled up \$200 billion dollars over the last 50 years, we would have still come out ahead by 2000.


But the more interesting issue is whether there are other Y2K-type problems we should be aware of. How many more “new riddles” are there? The Y2K problem, Unix’s “elapsed time since 1/1/1970” problem, and others of this ilk are the residue

of faulty introspection in our software engineering efforts. Perhaps detection models for predicate projectability and sundry other new riddles of inductive reasoning should be added to our working stock of software metrics. According to Jones, the “optimal time” to begin Y2K repairs was 1995 or earlier, with October 1997 as the last point that a mid-size corporation could commence their repairs with any hope of finishing by 2000. Perhaps the time to look for these detectable, though not-yet-detected, new riddles is now.

### For Further Reading

- Jones, C. *The Year 2000 Software Problem: Quantifying the Costs and Assessing the Consequences*, ACM Press, New York, 1997. The appendices contain lists of a variety of useful

Web, print, and consulting resources dealing with the Y2K problem.

- Wegner, P. “Why Interaction is More Powerful than Algorithms” (*Communications*, May 1997, pp. 80–91). My response appeared in the September 1997 “Forum,” pp. 20–21. 

---

HAL BERGHEL ([www.acm.org/~hbl](http://www.acm.org/~hbl)) is a professor of computer science at the University of Arkansas and a frequent contributor to the literature on cyberspace.

---