On the Avoidance of Error

Roland Backhouse

July 2025

1 Introduction

I believe that I first met Johan when he was an undergraduate in Groningen but I did not get to know him well until after he had completed his PhD thesis [Jeu93]. The first paragraph in his thesis is about the frustration he experienced as the result of an error in a mailer. When asked to submit an article for his festschrift my immediate thought was therefore to write something about the errors that I have encountered, and to include some brief comments about how to avoid error.

There are lots of examples I could choose but, to keep it brief, I have limited the examples to two from my early career (in sections 2 and 3) and one quite recent example (section 4). The final section draws some conclusions on how to avoid error.

2 Conway's Factor Matrix

One of the first errors in the literature I remember very clearly is in Conway's book [Con71]. I was attracted to the book because of the word "Algebra" in the title; at the time, and still today, I felt that algebra was of vital importance to algorithm design, and regular algebra particularly so since it is the algebra of three components of all non-trivial algorithms: choice, sequencing and iteration. Conway's book had a great influence on my PhD study; the importance of algebra is clearly a sentiment shared by Johan.

I was particularly impressed by Conway's theory of *factors* of a language. Theorem 4 of chapter six introduces the *factor matrix* of a regular language E. Denoting the entries of the matrix by E_{ij}, and the left and right factors of E by L_i and R_i, the theorem is as follows.

> Each E_{ij} is a factor, and each factor is one of the E_{ij}. There exist unique indices l, r such that $E = L_r = R_l = E_{ij}$ and $L_i = E_{li}$ and $R_i = E_{ir}$ for each i. Hence the factors naturally form a square matrix among the entries of which is E.

Immediately following the proof of this theorem, the following note is made:

> This organization of the factors as E_{ij} does prevent E from occurring twice in its factor matrix,

Unfortunately, there is an error in this sentence. Instead of saying "does prevent" the sentence should say "does **not** prevent": there is a missing negation in the sentence!

I recall that I spent almost two days trying to understand why the theorem "does prevent E from occurring twice" before I changed tack and looked for an example of a language that *does* occur twice in its factor matrix. Indeed, such an example is easy to find. The language $(aa)^*$ has (admissible[1]) factor matrix

$$\begin{bmatrix} (aa)^* & (aa)^*a \\ (aa)^*a & (aa)^* \end{bmatrix} .$$

There are two distinct entries in this matrix, $(aa)^*$ and $(aa)^*a$ which both occur twice. Both are factors, and left factors, and right factors, of $(aa)^*$. So one is left wondering which are the "unique" indices l and r mentioned in the theorem! There are indeed two options — the only requirement is that $l=r$ (because $(aa)^*$ is the repeated entry on the diagonal).

In retrospect, it shouldn't have taken me two days to spot the missing "not" in Conway's note. Without the "not" the sentence sounds strange — at least to a native English speaker like myself — and the sentence ends

> and in general certain factors appear repeatedly.

How I missed this I do not know. However, this simple error highlights a problem with Conway's statement of the theorem. The theorem is correct so long as it is properly interpreted. The indices mentioned in the theorem are not arbitrary: in formal terms, the index given to the left factors is a function and the index given to the right factors is also a function, but these two functions must satisfy a couple of properties, one of which states how they are linked. The theorem assumes that these two functions are given; the claimed unicity properties are with respect to the given functions. (In fact, at least in my view, the index functions are an unnecessary complication: the theorem can be formulated in a way that avoids indexing the left and right factors.)

[1] In almost all cases, both $\emptyset$ and T^* are factors and left and right factors of a language E over the alphabet T; in this case, the factor matrix is 4×4, and not 2×2, with entries $\emptyset$ and a^* appearing repeatedly. In practical applications these entries are deemed "inadmissible" in the sense that they can be ignored. They have been omitted here for simplicity.

3 Quadratic Collision Handling

Another example of an error that took me a long time to resolve was concerned with hashing techniques. In my first lecturing post in the 1970s, I had to lecture on data structures. Since I had not previously studied hashing techniques, I learnt about them by studying the textbook that had been recommended by my predecessor on the course.

With regard to quadratic collision handling, the following assertion was made:

> When the table size is a power of two, only half the table is searched.

My immediate thought was that 1 is a power of two, so how could only half the table be searched in this case? The textbook, the authors and title of which I have now forgotten, provided no further information so I was obliged to look elsewhere. It then transpired that the assertion was a widely-held myth in the literature at the time, which originated in the first publication on the algorithm and then was repeated without questioning in several textbooks. I even found a "proof" of the assertion.

Quadratic collision handling is an algorithm for finding an empty location in a finite table. A hashing technique is used to find an initial location; if this location is full then a search begins. The first so-called "probe" is 1 place on from the initial location; if that location is full, the 2nd probe is 2 places on, and so on. On the ith iteration of the search, the probe is i places on from the last location. This means that, on the ith iteration, the probe is $1+2+\ldots+i$ locations on from the initial location. All of these calculations are made modulo the table size but, algebraically, that is insignificant.

If the table size is 2, it is obvious that both locations are probed. If the table size is 4, and the initial location is 0, quadratic collision handling searches in order locations $0+1$, $0+1+2$, and $0+1+2+3$ (all modulo 4); that is, including the initial location, the locations searched are 0, 1, 3 and 2. Thus, all locations in the table are searched.

The myth that only half the table is searched comes from a "generalisation" of the algorithm. Note that on the ith iteration, the location that is inspected is $\frac{1}{2}i+\frac{1}{2}i^2$ locations on from the initial location. The "generalisation" that was commonly made was to replace the constant coefficients by variables, a and b say, and then to implicitly assume that a and b are integers!

At the time (1973 or 1974) I found several publications that perpetuated the myth. Unfortunately, I can't recall which publications they were, nor where I found the "proof"; it is also likely that all of them are now out of print and unobtainable. Nevertheless, there is still evidence of the existence of the myth. Exercise 10.1.9 in [AU73] is the following:

> Show that if $h_i = \left[h_0 + ai^2 + bi\right] \bmod n$ for $1 \le i \le n-1$, then at most half the locations in the sequence h_0, h_1, h_2, $\ldots$, h_{n-1} are distinct.

The exercise says nothing about the type of the coefficients a and b but, for the exercise to be correct, one must assume that they are integers and $a \neq 0$.

Contrast this with the discussion of Quadratic Probing in a more recent textbook [CLRS09]:

> Quadratic probing uses a hash function of the form
>
> $$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m \ , \qquad (12.5)$$
>
> where (as in linear probing) h' is an auxiliary hash function, c_1 and $c_2 \neq 0$ are auxiliary constants, and $i = 0, 1, \ldots, m-1$.

Again, no type information is given about the coefficients c_1 and c_2. However, in problem 12-4, they give pseuo-code for the collision-handling algorithm I presented earlier and then give the following exercise.

> **a** Show that this scheme is an instance of the general "quadratic probing" scheme by exhibiting the appropriate constants c_1 and c_2 for equation (12.5).
>
> **b** Prove that this algorithm examines every table position in the worst case.

4 A Recent Example

A crticism that might be levelled at the examples in sections 2 and 3 is that they are from so long ago —50 years, in fact— that they are no longer relevant; modern technology, particularly automated verification, means that such errors are inconceivable. Or are they? The example in this section dates from 2022, (at the time of writing) less than three years ago.

In May 2022, I attended a meeting at Huawei in Cambridge at which the current state of the art in automated reasoning was being discussed. Among the techniques under scrutiny was so-called "neural learning", as espoused by OpenAI.

The OpenAI website boasted that "our models and search procedure are capable of producing proofs that chain multiple non-trivial reasoning steps". An example was given. See the screenshot shown in fig. 1 at the end of this document. (The webpage shown in fig. 1 is no longer accessible.)

The example is about a claimed property of two linear functions f and g that are linked by a constraint on the coefficients in their definition. The statement of the problem is first given in traditional mathematical notation following which the problem is restated in OpenAI-speak (the language used by the OpenAI system).

On looking at this webpage, a very bright young student (whose name I do not know) immediately pointed out that a simple typographical error had been made in the

assumptions: in the process of transliteration, "f" has been mistakingly written instead of "g". As a result, the assumptions labelled h_0, h_1 and h_2 are contradictory (see fig. 1), and anything you like can be deduced from them. Apparently, the system did not have the "intelligence" to spot such a simple mistake!

5 Avoiding Error

Over the years, I have spotted a number of errors in published papers, but the ones I spotted at the start of my career have had the most impact on me and are the ones I remember the best. Of course, we all make errors so it is important that we have ways of detecting them or, better still, avoiding them altogether.

During these years, automated verification systems have assumed much prominence in the literature. One wonders whether their use might have avoided the errors detailed in sections 2 and 3 above? The error in the OpenAI example (section 4) demonstrates that this is unlikely. The problem is that automated verification systems are too often used as rubber stamps. It is commonplace nowadays to read that the results in a paper have been verified by one or other verification system. But typically that verification involves a transliteration from one language to another; what is verified is not the same as what is asserted and there is no formal link between the two. As a result, it is quite likely that existing errors are overlooked or errors are introduced in the transliteration process.

Quite rightly, (post-hoc) verification of computer software was not the subject of Johan's thesis; rather, his thesis was about Constructive Algorithmics: an iterative process of deriving programs from their specifications using algebraic manipulation. As argued by Knuth [Knu84], programming is best viewed as a document-preparation activity, the documentation serving to integrate the many different aspects (requirements, specification, implementation, testing etc.) of a highly complex process. Furthermore the language of programming specification is the language of mathematics — in other words, precise and concise, but unconstrained and subject to continual evolution and adaptation.

As argued in [VB99], if automated systems are to be used effectively for the avoidance of error, it is important that their use involves a close dialogue during the documentation process. Moreover, there needs to be a clear and verifiable link between what is being documented and what is being verified. Unless such tools are used at all stages, errors will continue to be made and reported on well into the future.

References

[AU73] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation and compiling*, volume 2 of *Series in Automatic Computation*. Prentice-Hall, 1973.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd edition*. MIT Electrical Engineering and Computer Science Series, MIT Press, 2009.

[Con71] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[Jeu93] Johan Jeuring. *Theories for Algorithm Calculation*. PhD thesis, Rijksuniversiteit te Utrecht, 1993.

[Knu84] D.E. Knuth. Literate programming. *Computer Journal*, 27(2):97–111, 1984.

[VB99] Richard Verhoeven and Roland Backhouse. Interfacing program construction and verification. In Jeanette M. Wing, Jim Woodcock, and Jim Davies, editors, *World Congress on Formal Methods, FM '99*, number 1709 in LNCS, pages 1128–1146. Springer-Verlag, 1999.

We also observe that our models and search procedure are capable of producing proofs that chain multiple non-trivial reasoning steps. In the proof below, the model starts by using contraposition leading to the existential statement `(∃ (x : ℝ), f x ≠ a * x + b)`. It then generates a witness for it with `use (0 : ℝ)` and finishes the proof by leveraging the `norm_num` tactic.

Our models, trained with *statement curriculum learning*, were able to close a variety of problems from training textbooks as well as AMC12 and AIME competitions, and 2 problems adapted from the IMO. We present below three examples of such generated proofs.

Figure 1: Screenshot of OpenAI website taken on 9th May 2022