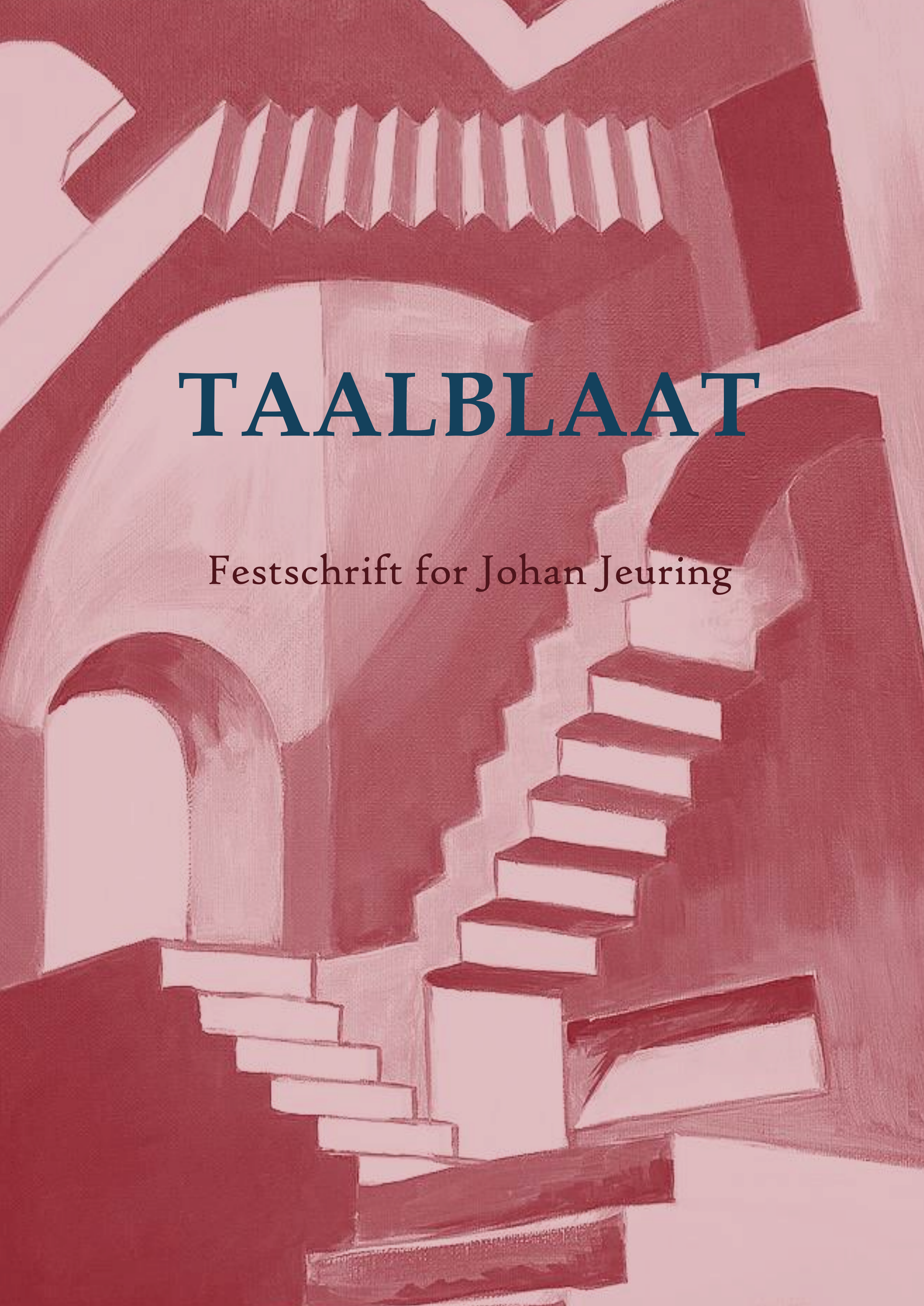
TAALBLAAT

Festschrift for Johan Jeuring



Taalblaat

Festschrift for Johan Jeuring

Preface

This Festschrift celebrates Johan Jeuring's 60th birthday and honors not only his contributions to computer science and education, but also the influence he has had on his friends and colleagues. Throughout his career, Johan has affected all of us in how we think about programming, education, and the intersection of theory and practice.

The articles collected in this Festschrift reflect the breadth and depth of this influence. They are wonderfully varied in both style and content, ranging from personal anecdotes of Johan's mentorship to rigorous mathematical proofs. Some authors share memories of collaboration and friendship, while others present new research inspired by Johan's work in functional programming, generic programming, and educational technology.

The cover image is inspired by the cover of Johan's PhD thesis, which featured a staircase in black and white. The staircases on this cover represent the various research directions he has pursued throughout his career, which sometimes came together again. Regarding the color, many people could probably guess that Johan's favorite color is red. Lastly, the palindrome title reflects one of Johan's many interests.

We hope that Johan will enjoy reading this Festschrift as much as we have enjoyed creating it. Each contribution reflects not only academic respect but also genuine appreciation for a colleague, mentor, and friend.

July 2025

Alex Gerdes
Hieke Keuning
Wouter Swierstra

Table of Contents

On the Avoidance of Error

Roland Backhouse[0000-0002-0140-8089]

1 Introduction

I first met Johan Jeuring 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 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] on Regular Algebra and Finite Machines. 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

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

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, the index functions are an unnecessary complication: the theorem can be formulated in a way that avoids them altogether.)

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.

[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.

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 widespread myth in the literature at the time, that 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 $h_i = \left[h_0 + ai^2 + bi\right] \mathsf{mod}\ n$ for $1 \leq i \leq n-1$, 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
> $$h(k,i) = (h'(k) + c_1 i + c_2 i^2)\ \mathsf{mod}\ m, \qquad (12.5)$$
> 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 criticism 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 h0, h1 and h2 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 necessarily 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 like the ones exemplified above 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.

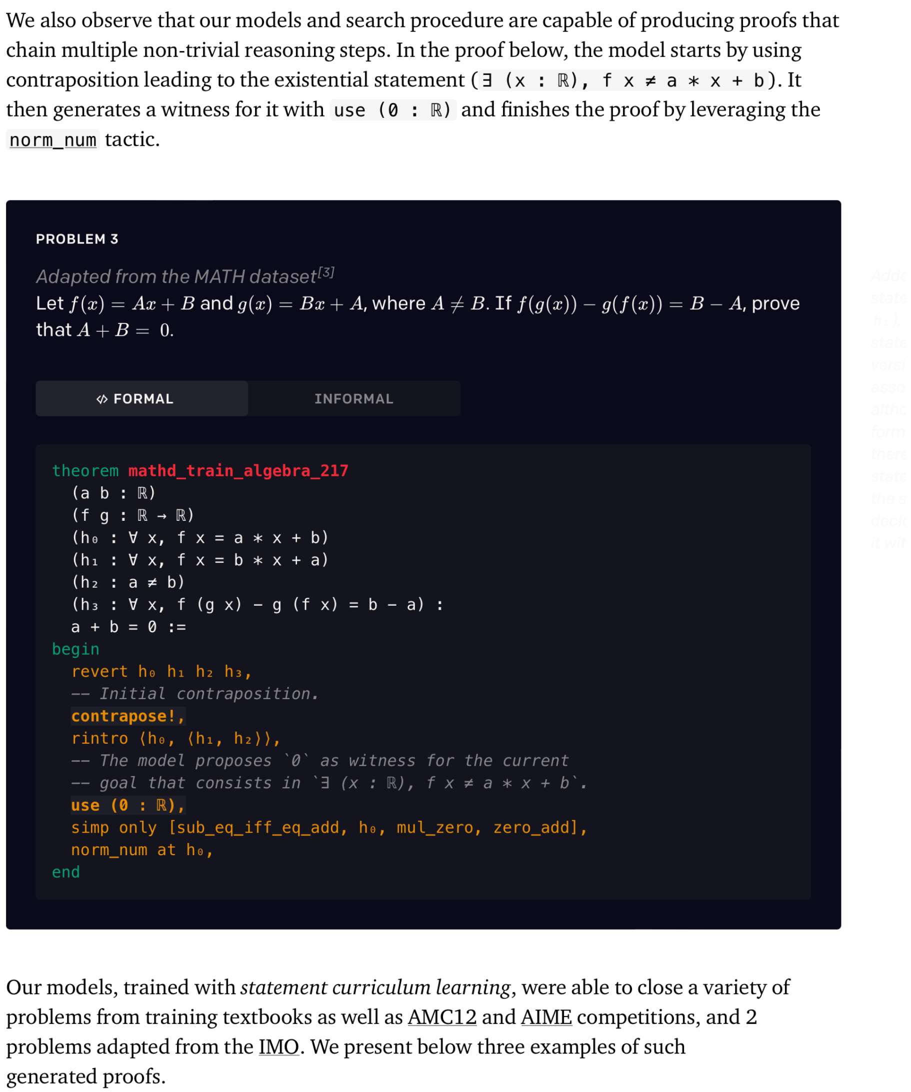
CLRS09. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd edition.* MIT Electrical Engineering and Computer Science Series, MIT Press, 2009.

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.

PROBLEM 3

Adapted from the MATH dataset[3]

Let $f(x) = Ax + B$ and $g(x) = Bx + A$, where $A \neq B$. If $f(g(x)) - g(f(x)) = B - A$, prove that $A + B = 0$.

FORMAL INFORMAL

```
theorem mathd_train_algebra_217
  (a b : ℝ)
  (f g : ℝ → ℝ)
  (h₀ : ∀ x, f x = a * x + b)
  (h₁ : ∀ x, f x = b * x + a)
  (h₂ : a ≠ b)
  (h₃ : ∀ x, f (g x) - g (f x) = b - a) :
  a + b = 0 :=
begin
  revert h₀ h₁ h₂ h₃,
  -- Initial contraposition.
  contrapose!,
  rintro ⟨h₀, ⟨h₁, h₂⟩⟩,
  -- The model proposes `0` as witness for the current
  -- goal that consists in `∃ (x : ℝ), f x ≠ a * x + b`.
  use (0 : ℝ),
  simp only [sub_eq_iff_eq_add, h₀, mul_zero, zero_add],
  norm_num at h₀,
end
```

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.

Fig. 1. Screenshot of OpenAI website taken on 9th May 2022

Stepwise refinement with Dafny assistance

Lex Bijlsma

1 Introduction

Dafny [2] is an automatic verification system for programs using pre- and post-conditions, invariants and variant functions. The exercises in a recent textbook [3] show that the use of this system is simple enough for students embarking on program derivation. It eliminates the practical problem of students' calculational skill being inadequate for doing predicate calculus by hand.

In this paper we demonstrate that the program construction style known as stepwise refinement [4] can also profit from assistance from the Dafny verifier. The advantage of enlisting Dafny will be that at all stages of the refinement process we can be confident that we have made no mistakes and that the structure we have so far arrived at will indeed produce a correct working program if we succeed in filling in the gaps that have been specified but not yet implemented.

For demonstration purposes, we shall return to the example from the original paper [4]. The problem is stated there as follows:

> Given are an 8×8 chessboard and 8 queens which are hostile to each other. Find a position for each queen (a configuration) such that no queen may be taken by any other queen (i.e. such that every row, column, and diagonal contains at most one queen).

The talk of chessboards and hostile queens seems intended to aid the reader in visualising the problem; however, it does not have much to do with the actual rules of chess. The game cannot produce a situation where more than two queens are present; moreover, in chess a queen can only take another when they are of different color.

If we are to verify the steps taken, we run up against the problem that it is not a priori clear that the configuration aimed for actually exists. Indeed, this is only found out through running the finished program. As explained clearly by Dijkstra [1], the only thing we can do is to generate all configurations where no two queens threaten each other, and check whether any of these do accommodate as many as eight queens.

2 First version

The queens in the stated problem seem to be without any color and indeed have no properties except the square where they are located. We therefore introduce a type `Coordinates` that specify the row and column of such a square, as follows:

```
datatype Position = Coordinates(nat, nat)
```

The only meaningful positions are those within the chessboard, i.e. with coordinates in the range 1..8. To express this, we introduce the predicate `Valid` as follows:

```
predicate Valid(pos: Position)
{
  match pos
  case Coordinates(hor, vert) ⇒
    1 ≤ hor ≤ 8 ∧ 1 ≤ vert ≤ 8
}
```

Next we have to state what it means that one queen is in a position to take another, that is to say that they share a row, column or diagonal:

```
predicate Threatens(queen: Position, pos: Position)
{
  match (queen, pos)
  case (Coordinates(qx, qy), Coordinates(px, py)) ⇒
  qx = px ∨ qy = py ∨ qx-qy = px-py ∨ qx+qy = px+py
}
```

Here we have the complete apparatus to express the goal of the exercise:

```
method PlaceAllQueens() returns (queens: set<Position>)
ensures ∀ queen: Position | queen ∈ queens :: Valid(queen)
ensures ∀ queen1: Position, queen2: Position |
  queen1 ∈ queens ∧ queen2 ∈ queens ∧ queen1 ≠ queen2 ::
  ¬Threatens(queen1, queen2)
```

We can submit this to the Dafny verifier and it is accepted. But this only reflects the well-formedness of the specification: if we had omitted the conjunct `queen1 ≠ queen2` in the domain of the second postcondition, verification would still succeed but it would have been impossible to satisfy the specification in a non-trivial way (because the definition of `Threatens` implies that every queen threatens itself – yet another discrepancy with the rules of chess).

3 Second version

It follows immediately that every column on the chessboard can only contain one unthreatened queen. We therefore decide to fill up the board column by column: for every vacant column, we try to find a suitable unthreatened space to place the next queen. This gives rise to the following implementation of `PlaceAllQueens`:

```
method PlaceAllQueens() returns (queens: set<Position>)
ensures ∀ queen: Position | queen ∈ queens :: Valid(queen)
ensures ∀ queen1: Position, queen2: Position |
  queen1 ∈ queens ∧ queen2 ∈ queens ∧ queen1 ≠ queen2 ::
  ¬Threatens(queen1, queen2)
```

```
{
  queens := {};
  var columnsSearched := 0;
  while columnsSearched < 8
  decreases 9 - columnsSearched
  invariant ∀ queen: Position | queen ∈ queens :: Valid(queen)
  invariant ∀ queen1: Position, queen2: Position |
    queen1 ∈ queens ∧ queen2 ∈ queens ∧ queen1 ≠ queen2 ::
    ¬Threatens(queen1, queen2)
  {
    var found: Position;
    found := NewSafePosition(queens, columnsSearched);
    assert columnsSearched < Column(found);

    if Valid(found) {
      assert IsSafePosition(queens, found);
      assert ∀ queen: Position | queen ∈ queens ::
        ¬Threatens(queen, found);
      queens := queens + {found};
    }
    columnsSearched := Column(found);
  }
}
```

For convenience' sake, we have introduced the abbreviation

```
function Column(pos: Position): nat
{
  match pos
  case Coordinates(hor, vert) ⇒ hor
}
```

More importantly, we have postponed both the test of whether a position is safe and the task of finding one to the next refinement step, merely specifying:

```
predicate IsSafePosition(queens: set<Position>, pos: Position)
requires ∀ k | k ∈ queens :: Valid(k)
ensures IsSafePosition(queens, pos) = ∀ k | k ∈ queens ::
  ¬Threatens(k, pos)

method NewSafePosition(queens: set<Position>,
                       columnsSearched: nat)
  returns (found: Position)
requires ∀ k | k ∈ queens :: Valid(k)
requires 0 ≤ columnsSearched < 8
ensures Valid(found) ⟹ IsSafePosition(queens, found)
ensures columnsSearched < Column(found)
```

Again, all is verified. Again this indicates mere well-formedness for the last specifications, but this time it also tells us the columnwise approach to the implementation of `PlaceAllQueens` will succeed if we manage to come up with the next refinements.

4 Third version

Let us start with the safety test, as that is entirely straightforward: merely a trivial copy of the specification!

```
predicate IsSafePosition(queens: set<Position>, pos: Position)
requires ∀ k | k ∈ queens :: Valid(k)
ensures IsSafePosition(queens, pos) = ∀ k | k ∈ queens ::
  ¬Threatens(k, pos)
{
  ∀ k | k ∈ queens :: ¬Threatens(k, pos)
}
```

But finding a safe position requires more work, although it is only a matter of sequentially checking squares.

```
method NewSafePosition(queens: set<Position>,
                       columnsSearched: nat)
  returns (found: Position)
requires ∀ k | k ∈ queens :: Valid(k)
requires 0 ≤ columnsSearched < 8
ensures Valid(found) ⟹ IsSafePosition(queens, found)
ensures columnsSearched < Column(found)
{
  var nextcolumn := columnsSearched + 1;
  while nextcolumn ≤ 8
  invariant columnsSearched < nextcolumn ≤ 9
  decreases 9 - nextcolumn
  {
    var nextrow := 1; // arbitrary
    while nextrow ≤ 8
    decreases 9 - nextrow
    {
      var nextTry := Coordinates(nextcolumn, nextrow);
      if IsSafePosition(queens, nextTry)
      { found := nextTry;
        return;
      }
      else {nextrow := nextrow + 1;}
    }
    if nextcolumn ≠ 9 {nextcolumn := nextcolumn + 1;}
  }
  // if nextcolumn = 9
  {found := Coordinates(9, 9);}
```

```
}
```

The assignment `found := Coordinates(9, 9);` is only there to signal the absence of a solution in the remaining columns (the value 9 acting as a sentinel). Again, this verifies and there are now no unimplemented specifications left. The initialization of `nextrow` in the inner loop is arbitrary; for finding all safe configurations other possibilities here should be tried as well.

5 Testing

The Dafny verifier ensures that method `PlaceAllQueens` returns a set of positions that is safe, i.e. mutually non-threatening. What it does not guarantee is that this set's cardinality will be 8, in other words, that we have succeeded in placing a queen in every column. So let's run the program and see what results. The positions found by the current version are

$$(1, 1), (2, 3), (3, 5), (4, 2), (5, 4)$$

But after these five placements we are stuck: obviously the queen for column 6 cannot be placed in any of the rows 1..5 (threatened horizontally); but (6, 6) is threatened diagonally by (1,1), (6,7) by (2, 3) and (6,8) by (3, 5). Clearly the number of queens placed is much dependent on the order in which the safe places are selected, and the Dafny specifications offer no help here. The classical treatment of the problem [4] does not have any suggestions on the most advantageous order either, except trying everything by recursive backtracking. And even that approach does not enable us to assert a postcondition `|queens| = 8`, for we do not know a priori that such a solution exists.

6 Conclusion

If we compare this process to the presentation originally given [4], there are two major differences. In the first place, the original treatment does not specify the steps postponed to the next refinement, but describes their actions in a kind of informal pseudocode – which does not lead to a reliable proof. Dafny, on the other hand, provides a proved program. In the second place, traditionally there is no feedback during the process, and feedback can only be provided by testing once the code is complete. Dafny, on the other hand, checks whether the use of as yet unimplemented steps is consistent with their specifications. In both these aspects, the assistance provided by Dafny is a big help in preventing mistakes. However, the specifications offer no help in finding the most productive order in which to consider possible placements, nor in proving that an eight-queen solution exists.

References

1. Dijkstra, E.W.: EWD316: A short introduction to the art of programming (1971), circulated privately, available from the EWD Archive at the University of Texas
2. Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: Clarke, E., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, pp. 348–370. Springer (2010)
3. Leino, K.R.M.: Program Proofs. MIT Press (2023)
4. Wirth, N.: Program development by stepwise refinement. Communications of the ACM **14**, 221–227 (1971)

“Nondeju, gij hoort HOERA”

Maarten Fokkinga

→ ↑ ← ← ↑ → → → ↓ ← ↓ ↑ ↑ ← ↓ → → ↓ ↓ ← ← ↑ ↑ → ↑
- u u - j n i e u - u r i n j r - g g J t r j n r

↑ → → → ↓ ← ↑ ← ↑ ← ↓ ↓ ← ↑ ↑ → ↓ ↓ ← ↑ → ↑ → ← ↓
j i e u g - e n u n e t r j i e t t i i - g - n u

→ ↑ ← ↓ ← ↓ ← ↑ → ↑ ↑ → → ↓ ↓ ↓ ← ← ← ↑ ↑ ← ↑ ↑ →
g n u g - j - t r i t g n n - J i i r j e r t g n

→ → → ↓ ↓ ↓ ← ← ↑ ↑ → ↓ ↓ ↓ ↓ ← ← ← ↑ ↑ → → ↑ ↑ ←
n u u - J i r t g g J i i r r t g j e e J i i - u

← ← ← ↓ → → → → ↑ ↑ ← ←
n u n J i i r r - n n J

1 HOERA

Beste Johan, op 12 augustus 2025 heb ik luidkeels Hiep Hiep Hoera gezongen. Dat geluid had je vast niet verwacht uit het verre oosten, maar je *had* het het kunnen horen als je goed geluisterd had! Proficiat met je zestigste verjaardag.

Het is alweer lang geleden dat wij gedurende drie jaar op één kamer zaten. Beiden werkten we aan onze dissertatie. Jij als jong broekie, net afgestudeerd, en ik al in het midden van mijn loopbaan. Ik heb toen geen leeftijdsverschil tussen ons ervaren, terwijl ik nu al bijna twaalf jaar pensionado ben en jij nog zeven jaar “mag” voordat je de AOW leeftijd bereikt. Wat hebben we daar zitten kriebelen (squiggling). En met succes. Een HOERA was ook na afloop van die periode duidelijk te horen: voltooing van onze proefschriften. Ik denk met véél plezier terug aan die tijd.

2 HOERA

Bij een verjaardag hoort een presentje. Deze puzzel wil ik je graag geven:

Er zijn tien stukken `j t _ J e u r i n g` en twee lege plekken, omgeven door een vaste rand.
Een stuk kan verschoven worden (een “zet”) als er een passend lege plek naast zit. Stuk `J` is iets dunner dan de andere stukken, en kan door een gleuf midden onderin uit het bord geschoven worden. De opgave is in zo min mogelijk zetten het stuk `J` midden onderin te krijgen – en dan uit het bord te schuiven.

Misschien vind je het presentje helemaal niet leuk; je bent er dagen mee zoet en wat voor nuttigs heb je bereikt als stuk J er uit is? Mijn eigen pogingen mislukten; na een twintigtal zetten wist ik niet meer of ik in een situatie kwam die ik al eerder gehad heb. Spoiler: er zijn 112 zetten nodig.

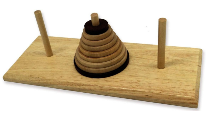
Maar –en nu hoor ik *jou* al Hoera roepen– de oplossing heb je al gezien!

3 HOERA

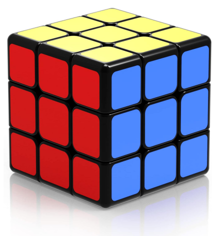
Ik heb me lang afgevraagd hoe de oplossing voor deze puzzel geprogrammeerd kan worden. Heb je daar zelf al een idee voor? Ik vond het zó verrassend eenvoudig dat ik, *nonjedu*, hardop Hoera riep. Dat heb je misschien wel gehoord!

Een programma dat een kortste pad in een graaf berekent, kan gebruikt worden om deze puzzel op te lossen. Een knoop in de graaf is dan een configuratie van de puzzel (de posities van de stukken op het bord), en een transitie in de graaf is een zet in de puzzel (het verschuiven van een stuk: een overgang van de ene configuratie naar een andere). De beginsituatie is de startknoop; iedere configuratie waarin stuk J midden onderaan staat, is een doelknoop. Omdat in de puzzel iedere zet omkeerbaar is, zijn de kanten in de graaf ongericht, en omdat iedere zet even zwaar telt, zijn de kanten gewichtloos. Een kortste pad in deze graaf, van de startknoop naar een doelknoop, geeft de oplossing voor de puzzel: de kortste zettenreeks van beginconfiguratie naar een doelconfiguratie! Een verschil met gangbare formuleringen van "het kortste pad in een graaf" is dat de knopen van de graaf niet bij voorbaat gegeven zijn, maar gedurende de berekening beetje bij beetje gegenereerd worden, steeds wanneer van een knoop de mogelijke zetten opgevraagd worden; dit impliceert dat de graaf samenhangend is. Ik wil je graag een Haskell programma hiervoor als presentje aanbieden.

Bovendien –hoera– kunnen *vele* puzzels met deze aanpak opgelost worden: al díe puzzels waarbij er sprake is van "zetten" die de ene "configuratie" overvoeren in een andere, en het de bedoeling is een kortste zettenreeks te vinden die van een gegeven beginconfiguratie gaat naar een gewenste eindconfiguratie. Denk bijvoorbeeld aan deze puzzels:







Torens van Hanoi, de '1-15 schuifpuzzel' waarin stukken met de getallen 1-15 via één lege plek op volgorde geschoven moeten worden, de puzzel waarbij auto's orthogonaal op een veld staan en alleen maar in hun lengterichting geschoven kunnen worden en "de rode auto" naar de uitgang moet, Rubik's Cube, en zo voorts.

In een streven naar een elegante formulering splits ik het probleem op in (1) het genereren van alle "kortste" paden en (2) het testen of een pad eindigt in een doelknoop. Lazy evaluation zal deze afzonderlijk geformuleerde berekeningen in de tijd verweven laten plaats vinden. Een "kortste" pad is: een pad dat in de graaf een kortste pad is van zijn startknoop naar zijn eindknoop.

Voor taak (2) ligt de functie voor de hand:

$$kortstePad = head.\ dropWhile \text{ padeinde-is-geen-doelknoop}$$

Voor de eenvoud maar even aangenomen dat er altijd een kortste pad bestaat.

Voor taak (1) voldoet de volgende functie f; die genereert bij lijst-argument [pad-met-alleen-*startKnoop*] een lijst van kortste paden vanuit *startKnoop* naar iedere knoop in de graaf. De definitie is verrassend eenvoudig:

$$\begin{aligned} &f\ [\,] &&= [\,] \\ &f\ (p{:}ps) &&= p : \ f\ (ps +\!\!+\ qs) \\ &\quad \text{waarbij} \\ &\quad qs = [\, p \text{ verlengd met } k' \mid k' \text{ is buur van eindknoop p die niet zit in } p{:}ps\,] \end{aligned}$$

Johan, zie je dat f een anamorfisme is? Ik krijg weer warme gevoelens.

Een operationele verklaring van de werking luidt als volgt. Na de initiële aanroep f[pad-met-alleen-*startKnoop*] geldt in iedere aanroep van f: het argument is een lijst van kortste paden vanuit *startKnoop* die ieder met nul of meer knopen uitgebreid gaan worden. De lengtes van deze paden verschillen hooguit één en de langere paden staan achteraan. Steeds wanneer een pad opgeleverd wordt, worden de 1-staps uitbreidingen daarvan achteraan in het argument van de volgende aanroep van f gezet. Het argument van f fungeert als een queue: het voorste pad p gaat eruit en de nieuwe paden qs komen achteraan. Functie f genereert de paden in een breadth-first manier.

Een formeel correctheidsbewijs zal gebaseerd zijn op deze eigenschap P:

$$P(f) \quad = \quad \forall ps. \quad A\ (ps) \Rightarrow B\ (ps, f\ ps)$$

waarbij

- $A(ps)$ = "ps is een lijst van kortste paden vanuit eenzelfde startknoop en met lengtes die hooguit één verschillen, gesorteeerd naar oplopende padlengte."
- $B(ps, rs)$ = "rs is een maximale lijst van paden die een (nul- of meerstaps) 'kortste uitbreiding' zijn van een pad in ps; lijst rs is oplopend gesorteerd naar padlengte."

De implicatie $A(ps) \Rightarrow B(p{:}ps,\ f(p{:}ps))$, gedaan over de linkerkant van de definitie, volgt uit de rechterkant van de definitie en de aanname (inductiehypothese) dat $P(f)$ geldt voor de recursieve aanroep, dat wil zeggen: de aanname $A(ps +\!\!+\ qs) \Rightarrow B(ps +\!\!+\ qs,\ f(ps +\!\!+\ qs))$. Dus met inductie naar het aantal unfold-stappen in de berekening, is $P(f)$ waar.

Van paden worden alleen maar eindknopen getest en toegevoegd; daarom representeer ik, binnen f, paden met de startknoop in de staart en de eindknoop aan kop:

$$f\ (p@(k:\_) : ps) = \mathit{reverse}\, p :\ f\ (ps +\!\!+ [k'{:}p \mid k' \leftarrow \mathit{buren}\, k,\ k\, \mathrm{nietIn}\, p{:}ps])$$

Toepassing. Om met f de puzzel op te lossen, moeten we ook de puzzel zelf representeren. Een formele representatie ligt voor de hand en zal ik hier schetsen. Een stuk heeft een naam, breedte en hoogte: (nm, b, h). Het bord heeft posities $0..B-1$ in de breedte en $0..H-1$ in de hoogte. Een configuratie geeft bij ieder stuk een positie: *type Configuratie* $= [(\mathit{Stuk}, \mathit{Positie})]$. Twee geplaatste stukken $((\_, b, h), (x, y))$ en $((\_, b', h'), (x', y'))$ zijn disjunct als $x+b \leq x' \vee x'+b' \leq x \vee y+h \leq y' \vee y'+h' \leq y$. Een geplaatst stuk $((nm, b, h), (x, y))$ zit binnenboord als $0 \leq x \leq B-b \wedge 0 \leq y \leq H-h$. Mogelijke zetten vanuit een configuratie c zijn als volgt te formaliseren (in $\mathit{zetten} :: \mathit{Configuratie} \rightarrow [\mathit{Configuratie}]$). Genereer nieuwe configuraties door íeder stuk van c één positie in één van de vier richtingen te verschuiven (dus 10×4 mogelijkheden), op voorwaarde dat het stuk nog binnenboord is en disjunct is van de andere stukken van c. Om de puzzel als graaf te zien kiezen we: $\mathit{Knoop} = \mathit{Configuratie}$, $\mathit{buren} = \mathit{zetten}$, $\mathit{startKnoop} = \mathit{beginConfiguratie}$, $\mathit{doelKnoop} = \mathit{doelConfiguratie}$.

Nondeju. Het duurt te lang voordat het programma een oplossing presenteert voor de `jt_Jeuring` schuifpuzzel.

4 Hoera

Gelukkig zijn er twee aanzienlijke optimalisaties mogelijk: één in de codering van functie f en één in de analyse van de puzzel en puzzels in het algemeen.

In de codering van f kan voor de test "nietIn $p{:}ps$" een aparte parameter *bezocht* aan f toegevoegd worden, en wel een *verzameling* (uit module `Data.Set`) van alle geproduceerde knopen. De test wordt dan vervangen door 'k niet in *bezocht*', en in de recursieve aanroep moet het extra argument een geschikte uitbreiding van *bezocht* zijn. Deze aanpassing reduceert de rekentijd aanzienlijk, met name door de efficiëntere datastructuur.

> **Terzijde.** In feite zijn van $p : ps$ hooguit de eindknopen en voorlaatste knopen van de paden nodig. Mijn experimenten suggereren dat deze optimalisatie nauwelijks de rekentijd reduceert maar de code wel compliceert.
>
> Diverse andere optimalisaties zijn nog denkbaar, met name de "promotion" van padeinde-is-geen-doelknoop uit *kortstePad* naar voren in de berekening, tot in f zelf. Ik heb geen optimalisatie gevonden die –voor de puzzel– de rekentijd echt reduceert.

Ten aanzien van puzzels in het algemeen realiseer ik me dat verschillende configuraties c en c' *equivalent* kunnen zijn in de zin dat een zettenreeks naar c en een naar c' allebei evenveel vervolgzetten nodig hebben om tot een gewenste kortste zettenreeks uitgebreid te worden. Voor de schuifpuzzel geldt dat twee configuraties equivalent zijn als ze op gelijke posities stukken van gelijke breedte en hoogte hebben; de vier vertikale stukken hebben gelijke breedte en hoogte en dat geldt ook voor de vier kleine vierkante stukken. Ook zijn twee configuraties equivalent als ze gespiegeld zijn langs de vertikale as door het midden.

De test 'k niet in *bezocht*' wordt dus vervangen door 'k is niet equivalent met een knoop in *bezocht*'. We kunnen dure equivalentietesten vervangen door goedkopere gelijkheistesten, door niet de configuraties zelf op te slaan in *bezocht* maar 'unieke representanten van de equivalentieklassen'. Zo'n representant noem ik in puzzeltermen: *abstracte configuratie*, en in graaftermen: *abstracte knoop*. Met name, voor de `jt_Jeuring` schuifpuzzel, geeft een abstracte configuratie bij iedere positie de breedte en hoogte aan, opgeslagen in een lijst gesorteerd naar positie. Twee configuraties zijn equivalent als hun abstracte configuraties gelijk zijn.

Met deze twee optimalisaties werkt het programma bevredigend. Voor de schuifpuzzel levert het de oplossing van 112 zetten die aan het begin staat; daarbij worden 23798 zettenreeksen berekend, 77378 configuraties en 23905 abstracte configuraties. Op mijn stokoude PC duurt dat minder dan 8 seconden.

Terzijde. Een reeks van 112 configuraties leest wat lastig. Daarom heb ik f iets uitgebreid door niet paden van knopen op te leveren maar paden van transities; een transitie bevat naast de 'nieuwe' knoop ook een indicatie van de stap die gedaan is. Functie $buren :: Knoop \rightarrow [Knoop]$ wordt dus vervangen door $transities :: Knoop \rightarrow [(Stap, Knoop)]$. De puzzelaar moet de functie *zetten* overeenkomstig aanpassen. In de uitkomst van f kan desgewenst de knoop-component verwijderd worden zodat alleen de stappen overblijven. Deze aanpassing heeft geen merkbare invloed op de rekentijd.

5 Een laatste Hoera

Tot slot nog een puzzel, speciaal voor jou, Johan. Hoera, hoef jij je het komend weekend niet te vervelen. Vind een kortste reeks plaatswisselingen van letters waarmee *start* getransformeerd wordt tot *eind*, waarbij:

$start = (map\ toLower .\ filter\ isLetter)$ "`Johan Theodoor Jeuring`"
$eind\ = (map\ toLower .\ filter\ isLetter)$ "`Nondeju, gij hoort Hoera`"

Vanuit iedere configuratie (letterreeks) zijn 190 verschillende zetten (letterwisselingen) mogelijk, namelijk $[(i, j) | i \leftarrow [0..19], j \leftarrow [i{+}1..19]]$. Als je dit met de hand oplost, win je het misschien wel van het programma, want op papier zie je zo wat zinvolle wisselingen zijn. Het programma levert de volgende reeks op, met lengte 14:

0-5 0-14 0-4 2-10 2-12 2-18 3-19 3-7 3-17 3-8 3-16 3-13 3-9 6-15

(Hieruit volgt dat 0-5-14-4 en 2-10-12-18 en 3-19-7-17-8-16-13-9 en 6-15 de cycles zijn van een permutatie van *start* naar *eind*.) Tijdens de berekening zijn er 361942 reeksen gevormd en 5966258 anagrammen (waarvan sommige al eerder "bezocht" waren), en bevat *bezocht* uiteindelijk 365108 anagrammen. De rekentijd was circa 15 minuten; al mijn pogingen om de meest voor de hand liggende lijst-representatie van letterreeksen (en de wisseling van letters!) efficiënter te maken gaven geen reductie van de rekentijd.

Johan, ik wens je nog een paar vruchtbare jaren toe in je loopbaan en thuis!

Maarten

```
import qualified Data.Set as S

-- ============== interface Puzzel-Graaf ============================

data Graaf knoop absKnoop stap = Graaf
   { transities  :: knoop -> [(stap, knoop)]
   , startKnoop  :: knoop
   , isDoelKnoop :: knoop -> Bool
   , mkAbsKnoop  :: knoop -> absKnoop
   }
kortstePad   :: Ord aknp => (Graaf knp aknp stp) -> [stp]
kortstePaden :: Ord aknp => (Graaf knp aknp stp) -> [[stp]]

-- ============== Kortste pad in een Graaf  =========================

-- Nomenclatuur: k: Knoop; a: AbsKnoop; s: Stap; t: Transitie; p,q: Pad,
-- postfix s voor meervoud.
-- Transitie = (Stap,Knoop);  Pad = [Transitie]
-- bf genereert alle kortste paden in een breadth-first volgorde.

-- kortstePad/Paden toont van elk pad alleen de stappen, door "map fst".
kortstePaden = map (map fst). bf
kortstePad graaf = (map fst. head. dropWhile notDoelBereikt. bf) graaf
   where
   notDoelBereikt [] = True
   notDoelBereikt p =  (not. isDoelKnoop. snd. last) p
   Graaf _ _ isDoelKnoop _ = graaf

bf :: Ord aknp => (Graaf knp aknp stp) -> [[(stp,knp)]]
bf graaf = bf' [[(undefined,startKnoop)]] S.empty
   where
   bf' [] _ = []
   bf' (p@((_,k):_): ps) bezocht
     = (tail. reverse) p:    bf' (ps++qs) (as `S.union` bezocht)
       -- tail: verwijdert de "half-undefined starttransitie"
       where
       ts = [t| t@(s,k)<-transities k, mkAbsKnoop k `S.notMember` bezocht]
       qs = map (:p) ts
       as = (S.fromList. map mkAbsKnoop. map snd) ts
   Graaf transities startKnoop isDoelKnoop mkAbsKnoop = graaf
```

Agda-ventures with PolyP

Jeremy Gibbons[1][0000-0002-8426-9917] and
Patrik Jansson[2][0000-0003-3078-1437]

[1] University of Oxford, UK
https://www.cs.ox.ac.uk/people/jeremy.gibbons/
[2] Chalmers University of Technology and University of Gothenburg, SE
https://patrikja.owlstown.net/

Abstract. Revisiting Johan Jeuring's PolyP 30 years on, we note that a special-purpose language is no longer needed: general-purpose dependently typed programming suffices. This is a text-based adventure from software archeology, via codes to universes. Happy 60th Birthday, Johan!

1 Introduction

Among Johan Jeuring's contributions to the world, not the least is his programming language PolyP, developed in a series of papers from 1995 to 2002. One of us was his first PhD student, and part of this endeavour.

PolyP was a research language designed for the purpose of exploring the notion of *polytypic programming*: programs that are parametrized by the shape of datatypes, so that one program can be applied to many different datatypes. In the first paper on the topic [7], Jeuring quotes the definition from Webster's dictionary:

> **poly·typ·ic** [ˌpä-lē-ˈti-pik], *adj.*: having or involving several different types

Other names for the same idea include 'structurally polymorphic', 'shape polymorphic, 'type parametric', 'generic', and 'datatype-generic'. Typical polytypic programming problems are structural: equality, matching, folding, mapping, traversal, encoding, printing, parsing, unification, and so on. A crucial criterion is the maintenance of strong static type safety; in contrast, approaches based on dynamic typing may be able to express the same programs, but cannot make the same static guarantees.

PolyP was implemented [5] as a preprocessor for Haskell, providing an additional polytypic construct that gets translated into ordinary Haskell. (The source code is available at GitHub [3]. The original revision history has been preserved, predating GitHub's birth by a decade.) The work on PolyP led to a grant from the Dutch research council NWO for the *Generic Haskell* project, running 2000–2004 [8], another preprocessor for Haskell, and then in turn to many different approaches to generic programming [1].

So the ideas involved in PolyP have been influential over the past thirty years or so. But they have also been superseded by developments in programming languages. In particular, what in 1995 required a domain-specific language and a special-purpose preprocessor can be achieved in 2025 by good old-fashioned programming. This has been enabled by the advances that have since been made in *dependent types*. Whilst this theory significantly predates PolyP, it is only recently that tools originally envisioned as supporting theorem proving and formalized mathematics have become plausible programming languages.

In this short paper, we summarize the key ideas behind polytypic programming, and show how they can now be captured directly in a dependently typed programming language. Any dependently typed language will do, but we will use Agda. Maybe we can entice you back, Johan? The water is much warmer these days!

2 Polytypic programming

The general idea with PolyP is that "a polytypic function can be viewed as a family of functions: one function for each datatype" [9], defined by induction over the structure of the datatype. So first one needs to settle on the universe of datatypes.

PolyP used *polynomial types*: sums and products of some basic types, such as booleans, integers, and the unit type. For recursive datatypes such as lists of booleans and trees of integers, it used *regular functors*: initial algebras for functors constructed from polynomial operations on a type parameter, closed under certain compositions (so that one recursive datatype can be used in the shape functor for another). And to accommodate polymorphic (container) datatypes too, it extended to *regular bifunctors*.

For example [9], the Haskell datatypes of lists and rose trees

$$\begin{aligned}&\mathbf{data}\ \mathit{List}\ a\ = \mathit{Nil} \mid \mathit{Cons}\ a\ (\mathit{List}\ a)\\ &\mathbf{data}\ \mathit{Rose}\ a = \mathit{Fork}\ a\ (\mathit{List}\ (\mathit{Rose}\ a))\end{aligned}$$

are the initial algebras respectively of the bifunctors written in PolyP as

$$\begin{aligned}&\mathit{FList}\ = () + \mathbf{Par} \times \mathbf{Rec}\\ &\mathit{FRose} = \mathbf{Par} \times (\mathit{List}\ @\ \mathbf{Rec})\end{aligned}$$

For *FList*, the bifunctor is a sum, with the unit type for the left summand; the right summand is the product of the datatype parameter (that is, the first bifunctor argument) and a recursive call (the second bifunctor argument). For *FRose*, the right factor is the composition of *List* and the recursive position.

Continuing the example from [9], inductive datatypes $\mathbf{Mu}\ F\ A$ for bifunctor F and element type A have a constructor and a destructor:

$$\begin{aligned}&\mathit{inn} :: f\ a\ (\mathbf{Mu}\ f\ a) \to \mathbf{Mu}\ f\ a\\ &\mathit{out} :: \mathbf{Mu}\ f\ a \to f\ a\ (\mathbf{Mu}\ f\ a)\end{aligned}$$

A polytypic 'map' function for inductive datatypes

$$\begin{array}{l} \mathit{pmap} \quad :: (a \to b) \to \mathbf{Mu}\, f\ a \to \mathbf{Mu}\, f\ b \\ \mathit{pmap}\ p = \mathit{inn} \cdot \mathit{fmap}\ p\ (\mathit{pmap}\ p) \cdot \mathit{out} \end{array}$$

is defined in terms of a polytypic *fmap* for regular bifunctors:

$$\begin{array}{l} \textbf{polytypic}\ \mathit{fmap} :: (a \to c) \to (b \to d) \to f\ a\ b \to f\ c\ d \\ \quad = \lambda\ p\ r \to \textbf{case}\ f\ \textbf{of} \\ \qquad f + g \quad \to \mathit{fmap}\ p\ r\ \text{-+-}\ \mathit{fmap}\ p\ r \\ \qquad () \qquad \to \mathit{id} \\ \qquad \textbf{Con}\ t \to \mathit{id} \\ \qquad f \times g \quad \to \mathit{fmap}\ p\ r\ \text{-}\!\times\!\text{-}\ \mathit{fmap}\ p\ r \\ \qquad d\ @\ g \quad \to \mathit{pmap}\ (\mathit{fmap}\ p\ r) \\ \qquad \textbf{Par} \quad \to p \\ \qquad \textbf{Rec} \quad \to r \end{array}$$

where (-+-) and (-×-) map over sums and products respectively.

Note that there is no *Functor* or *Bifunctor* type class constraint on f, requiring a separate instance declaration, as would typically be the case in Haskell. Rather, the definition of *fmap* is essentially the template by which GHC would automatically *derive* a functor instance.

3 Dependently typed programming

In PolyP, a polytypic definition like that of *fmap* specifies what code should be generated for a specific type: "the compiler generates instances from the definition of the polytypic function and the type in the context where it is used" [5]. This is more than mere text processing, because PolyP does take care to type check a polytypic definition, in the sense that no generated instance will yield a Haskell type error. Still, PolyP is essentially a standalone domain-specific language for polytypic definitions, which means that the full power of the target language Haskell is not available in polytypic code.

This is a price that need not be paid, provided one can find a single language expressive enough to encompass both the polytypic templates and the actual eventual code. Then a separate code generation phase is not required: it becomes "a small matter of programming" in the one language. It turns out that a dependently typed language like Agda [13] provides the expressivity needed: types (and operations on types, such as functors and bifunctors) are values too.

So what would in PolyP be a polytypic function parametrized by a functor becomes in Agda just a function with an argument. However, that argument can't literally be a type, or a functor. We can't work with the types themselves, because we can't analyse them: they would be black boxes, and we need to perform case analyses on them. Instead, we make separate *codes* for the types in the universe, and define the interpretation mapping codes to types. Codes *can* be analysed and manipulated, since they are just terms in an algebraic datatype.

As a simple introduction, let's consider the universe of types consisting of sums and products of the unit type, naturals, and booleans. We start with an algebraic datatype of codes for the types in the universe:

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{Code}_0 : \mathsf{Set}\ \mathsf{where} \\
\quad \mathsf{NatT}\ \mathsf{BoolT}\ \mathsf{UnitT} : \mathsf{Code}_0 \\
\quad \_{*}\_\ \ \_{+}\_ : \mathsf{Code}_0 \to \mathsf{Code}_0 \to \mathsf{Code}_0
\end{array}
$$

For example, here is the code for the sum of the unit type and the product of naturals and booleans (that is, what would be $\mathit{Maybe}\ (\mathit{Nat}, \mathit{Bool})$ in Haskell):

$$
\begin{array}{l}
\mathsf{MaybeNatBoolCode} : \mathsf{Code}_0 \\
\mathsf{MaybeNatBoolCode} = \mathsf{UnitT} + (\mathsf{NatT} * \mathsf{BoolT})
\end{array}
$$

We can then define the interpretation of codes as types:

$$
\begin{array}{lcl}
[\![\_]\!]_0 : \mathsf{Code}_0 \to \mathsf{Set} \\
[\![\,\mathsf{NatT}\,]\!]_0 & = & \mathsf{Nat} \\
[\![\,\mathsf{BoolT}\,]\!]_0 & = & \mathsf{Bool} \\
[\![\,\mathsf{UnitT}\,]\!]_0 & = & \top \\
[\![\,c * c'\,]\!]_0 & = & [\![\,c\,]\!]_0 \times [\![\,c'\,]\!]_0 \\
[\![\,c + c'\,]\!]_0 & = & \mathsf{Sum}\ [\![\,c\,]\!]_0\ [\![\,c'\,]\!]_0
\end{array}
$$

This interpretation is simply a straightforward function definition—we have exploited Agda's fancy mix-fix syntax, but we might as well have named the function something like "interp0". The definition is by induction over the structure of codes: interpretations of the three base type codes are given directly ("$\top$" denotes the unit type, with sole element tt), and interpretations of the product and sum code constructors given inductively ("$\times$" and "Sum" denote product and sum types respectively).

Finally, we can define a polytypic function over this universe of types. For example, here is the equality function: it takes the code for some type in the universe, and two elements of the interpretation of that code, and returns a boolean. For the three base cases (constant types), the comparison is delegated to type-specific operators; for the two inductive cases (product and sum), it is given inductively.

$$
\begin{array}{lllll}
\multicolumn{5}{l}{\mathsf{equal}_0 : \{\ c : \mathsf{Code}_0\ \} \to [\![\,c\,]\!]_0 \to [\![\,c\,]\!]_0 \to \mathsf{Bool}} \\
\mathsf{equal}_0\ \{\mathsf{NatT}\} & n & m & = & (n ==_N m) \\
\mathsf{equal}_0\ \{\mathsf{BoolT}\} & x & y & = & (x ==_B y) \\
\mathsf{equal}_0\ \{\mathsf{UnitT}\} & x & y & = & (x ==_U y) \\
\mathsf{equal}_0\ \{c * c'\} & (x\ ,\ x') & (y\ ,\ y') & = & \mathsf{equal}_0\ x\ y \wedge \mathsf{equal}_0\ x'\ y' \\
\mathsf{equal}_0\ \{c + c'\} & (\mathsf{inj}_1\ x) & (\mathsf{inj}_1\ y) & = & \mathsf{equal}_0\ x\ y \\
\mathsf{equal}_0\ \{c + c'\} & (\mathsf{inj}_2\ x') & (\mathsf{inj}_2\ y') & = & \mathsf{equal}_0\ x'\ y' \\
\mathsf{equal}_0\ \{c + c'\} & \_ & \_ & = & \mathsf{false}
\end{array}
$$

For example, the two elements $\mathsf{inj}_1\ \mathsf{tt}$ and $\mathsf{inj}_2\ (3\ ,\ \mathsf{false})$ in the interpretation of MaybeNatBoolCode are not equal; and indeed, the expression

```
equal₀ { MaybeNatBoolCode } (inj₁ tt) (inj₂ (3 , false))
```

normalizes to false. Note that the first argument to equal_0 is written in curly braces, marking it as *implicit*, since it can be inferred. In particular, it is omitted for the recursive calls, and not needed for the example either: we can write just equal_0 (inj_1 tt) (inj_2 (3 , false)).

4 The full story

So much for codes for types, and their interpretation as actual types. If we want to handle inductive datatypes as fixpoints, we also want codes for functors. And for polymorphic inductive datatypes, we want bifunctors. So here are three mutually recursive datatypes of codes for them:

```
mutual
  data Type : Set where
    NatTy BoolTy UnitTy : Type

  data Functor : Set where
    Fix : Bifunctor → Functor

  data Bifunctor : Set where
    _*_ _+_ : Bifunctor → Bifunctor → Bifunctor
    Const   : Type                  → Bifunctor
    _•_     : Functor   → Bifunctor → Bifunctor
    Par Rec :                         Bifunctor
```

In order to interpret codes for inductive datatypes, we need to define these:

```
{-# NO_POSITIVITY_CHECK #-}
data Mu (f : Set → Set) : Set where
  In : f (Mu f) → Mu f

out : { f : Set → Set } → Mu f → f (Mu f)
out (In xs) = xs
```

Not all functors induce inductive datatypes, so we have to turn off the check that Agda would otherwise insist on. Since we are modelling PolyP generating Haskell, we don't worry too much about the risk of non-termination.

We can now give the interpretations of the three kinds of code:

```
mutual
  [[_]]T : Type      →             Set
  [[_]]F : Functor   → Set →       Set
  [[_]]B : Bifunctor → Set → Set → Set

  [[ NatTy ]]T  = Nat
  [[ BoolTy ]]T = Bool
```

```
⟦ UnitTy ⟧T  = ⊤

⟦ Fix f ⟧F p = Mu (⟦ f ⟧B p)

⟦ f * g ⟧B     p r = ⟦ f ⟧B p r × ⟦ g ⟧B p r
⟦ f + g ⟧B     p r = Sum (⟦ f ⟧B p r) (⟦ g ⟧B p r)
⟦ Const t ⟧B   p r = ⟦ t ⟧T
⟦ d • f ⟧B     p r = ⟦ d ⟧F (⟦ f ⟧B p r)
⟦ Par ⟧B       p r = p
⟦ Rec ⟧B       p r = r
```

Each base code is interpreted as the corresponding base type. Our only code for a functor is for a polymorphic inductive datatype, which is interpreted accordingly, using the interpretation of its bifunctor parameter. Bifunctor codes for lifted product and sum of two bifunctors are interpreted using the standard constructors; the codes for a constant bifunctor and for the composition of a functor and a bifunctor ("@" in PolyP, which is reserved in Agda so written with a bullet here) are defined recursively; and the 'parameter' and 'recursive argument' are projections.

Next we can define the functorial action for functors and bifunctors (the *pmap* and *fmap* we saw above), mutually recursive with catamorphisms:

```
mutual
  {-# TERMINATING #-}
  pmap : (d : Functor)    → (a → b) → ⟦ d ⟧F a → ⟦ d ⟧F b
  fmap  : (f : Bifunctor)  → (a → b) → (c → d) → ⟦ f ⟧B a c → ⟦ f ⟧B b d
  cata  : (f : Bifunctor)  → (⟦ f ⟧B a b → b) → ⟦ Fix f ⟧F a → b

  cata f h (In xs)          = h (fmap f id (cata f h) xs)

  pmap (Fix f) g            = cata f (In ∘ fmap f g id)

  fmap (f * g) p r (x , y)  = (fmap f p r x , fmap g p r y)
  fmap (f + g) p r (inj1 x) = inj1 (fmap f p r x)
  fmap (f + g) p r (inj2 y) = inj2 (fmap g p r y)
  fmap (Const t) p r x      = x
  fmap (d • g) p r xs       = pmap d (fmap g p r) xs
  fmap Par p r              = p
  fmap Rec p r              = r
```

For example, the code ListF for the shape bifunctor for lists, the corresponding code ListC for its fixpoint, and the interpretation MyList of the latter as an actual functor are:

```
ListF  : Bifunctor
ListF  = Const UnitTy + (Par * Rec)

ListC  : Functor
ListC  = Fix ListF
```

```
MyList : Set → Set
MyList = ⟦ ListC ⟧F
```

We can define constructors for these lists:

```
nilList : MyList a
nilList = In (inj₁ tt)

consList : a → MyList a → MyList a
consList x xs = In ( inj₂ ( ( x , xs )))
```

and conversion functions from and to built-in lists:

```
toMyList : List a → MyList a
toMyList = foldr consList nilList

fromMyList : MyList a → List a
fromMyList = cata ListF alg where
  alg : ⟦ ListF ⟧B a (List a) → List a
  alg (inj₁ tt)          = []
  alg (inj₂ ( x , xs )) = x :: xs
```

One canonical example of a polytypic function on polymorphic container datatypes is to “crush” it [11], aggregating the elements using a monoid:

```
mutual
  crush : (a → a → a) → a → (d : Functor) → ⟦ d ⟧F a → a
  crush  _⊕_ e (Fix f) = cata f (crushB _⊕_ e f)

  crushB : (a → a → a) → a → (f : Bifunctor) → ⟦ f ⟧B a a → a
  crushB _⊕_ e (f ∗ g)     (x , y)  = crushB _⊕_ e f x ⊕ crushB _⊕_ e g y
  crushB _⊕_ e (f + g)     (inj₁ x) = crushB _⊕_ e f x
  crushB _⊕_ e (f + g)     (inj₂ y) = crushB _⊕_ e g y
  crushB _⊕_ e (Const t)  x         = e
  crushB _⊕_ e Par         p         = p
  crushB _⊕_ e Rec         r         = r
  crushB _⊕_ e (d • g) = crush _⊕_ e d ∘ pmap d (crushB _⊕_ e g)
```

The binary operator is used to combine the two aggregations in a product, and the unit value is used for constants. For instance, we can flatten a container to a list, by making every element a singleton list then crushing using the list monoid:

```
flatten : (d : Functor) → ⟦ d ⟧F a → List a
flatten d = crush _++_ [] d ∘ pmap d (λ x → [ x ])
```

5 Polytypic packing and unpacking

Let us now look at a more extended example: another canonical piece of the polytypism literature, namely polytypic *packing*. By this we mean encoding a value of arbitrary type as a bitstream, in such a way as to be able (given information also about the type) to decode the bitstream back to the original data.

One can think of this as simple-minded data compression. For simplicity, we will encode to lists of bits and ignore the possible refinement of packing the bits into words. We name the bits used to label left and right choices, and provide a case analysis for them:

```
leftBit rightBit : Bool
leftBit  = false
rightBit = true

caseBit : Bool → a → a → a
caseBit b x y = if b then y else x
```

the idea being that

```
b = caseBit b leftBit rightBit
```

We provide primitives toBits and fromBits to convert between natural numbers and lists of booleans (making the simplifying assumption that all numbers are distinct to bitWidth bits—we could be cleverer about this):

```
bitWidth : Nat
bitWidth = 4 - we keep it small for testing

toBits : Nat → List Bool
toBits n = reverse (go bitWidth n) where
  go : Nat → Nat → List Bool
  go zero n     = []
  go (suc m) n = let (q , r) = divMod2 n in r :: go m q

fromBits : List Bool → Nat
fromBits = foldl (λ n b → (2 *N n) +N (if b then 1 else 0)) 0
```

Now, as a first attempt we might represent a packer for data as a function from that data to lists of bits. However, an unpacker would have to be more than simply a function in the opposite direction: we have to return the unused bits too, in order to be compositional; and we have to allow for failure, to make a total function. So we define the following:

```
Unpacker : Set → Set
Unpacker a = List Bool → Maybe (a × List Bool)
```

In fact, that type supports a monad—the combination of the state monad transformer on bit-list state around the maybe monad, what would in Haskell be written *StateT* [*Bool*] *Maybe*:

```
P : Set → Set
P = Unpacker
```

It turns out to be convenient to define packer using the same type:

```
Packer : Set → Set
Packer a = a → P ⊤
```

A packer will always succeed, and will produce rather than consume some bits. In that sense, the P monad is overkill—but it allows u now to think about composing packers and unpackers. Note that an unpacker Unpacker A for some type A is isomorphic to a function of type ⊤ → P A, which is nicely dual to the packer type A → P ⊤. (For more on this duality, see [6].)

The stateful interface is provided by two operations:

```
put : Packer (List Bool)
put bs' = λ bs → just (tt , bs')

get : Unpacker (List Bool)
get = λ bs → just (bs , bs)
```

But we will not need the full power of these two operations; we will use them only in restricted ways. For packers, we need only add some output:

```
packerTell : Packer (List Bool)
packerTell bs = do
  bs' ← get
  put (bs ++ bs')
```

(for reasons that will become clear in due course, we prepend rather than append). In particular, here are primitive packers for naturals, booleans, and the unit type:

```
packNat : Packer Nat
packNat n = packerTell (toBits n)

packBool : Packer Bool
packBool b = packerTell [ b ]

packUnit : Packer ⊤
packUnit tt = return tt
```

Note that packUnit is a no-op. Primitive unpackers for naturals and units are similarly easy:

```
unpackBool : Unpacker Bool
unpackBool = uncons

unpackUnit : Unpacker ⊤
unpackUnit = return tt
```

To unpack a natural, we read a chunk of input, then convert these bits to a number:

```
unpackNat : Unpacker Nat
unpackNat = do
```

```
xs ← get
let (ys , zs) = splitAt bitWidth xs
put zs
return (fromBits ys)
```

5.1 Packing as a monadic catamorphism

Now, to pack a structure of an inductive datatype, we will use a *monadic catamorphism* [12], which is like the ordinary catamorphism except that the algebra argument and the catamorphism itself are *Kleisli arrows*—that is, they have a monadic return type:

```
mutual
  cataM : (f : Bifunctor) → ([[ f ]]B a b → P b) → [[ Fix f ]]F a → P b
```

We will return to the definition of cataM shortly; but let's first see how it is used.

We define three mutually recursive functions, packers respectively for a type, a functor, and a bifunctor, taking correspondingly many element packers as arguments:

```
packT : (t : Type)      →                        Packer [[ t ]]T
packF : (d : Functor)   → Packer a →             Packer ([[ d ]]F a)
packB : (f : Bifunctor) → Packer a → Packer b → Packer ([[ f ]]B a b)
```

The packers for types each delegate to the appropriate primitive defined earlier:

```
packT NatTy n    = packNat n
packT BoolTy b   = packBool b
packT UnitTy tt  = packUnit tt
```

We only have one code for functors, interpreted as an inductive datatype, and this is where we use the monadic catamorphism:

```
packF (Fix f) p = cataM f (packB f p packUnit)
```

The catamorphism handles the recursive calls, so at the top level we do nothing (packUnit) for the recursive positions.

Finally, we have one fairly simple case per bifunctor:

```
packB (f ∗ g)   p q ( x , y ) = do packB g p q y
                                   packB f p q x
packB (f + g)   p q (inj₁ x)  = do packB f p q x
                                   packerTell [ leftBit ]
packB (f + g)   p q (inj₂ y)  = do packB g p q y
                                   packerTell [ rightBit ]
packB (Const t) p q           = packT t
```

```
packB (d • g)   p q     = packF d id ∘ pmap d (packB g p q)
packB Par       p q     = p
packB Rec       p q     = q
```

Recall that the primitive operations to write bits were defined to prepend to the list. We therefore specify that for products, we pack the right then the left component of the pair; then the left component will appear first in the output. Similarly—and more importantly—for sums, we emit the discriminator bit after packing the payload.

Now back to the monadic catamorphism. This requires a *distributive law* of the shape bifunctor over the monad—informally, this hoists the monad to the top, executing the computations for each of the recursive positions to make one composite computation collecting all the effects:

```
distr : (f : Bifunctor) → ⟦ f ⟧B a (P b) → P (⟦ f ⟧B a b)
```

Then the catamorphism deconstructs the data, makes recursive calls on each of the children, collects all their effects, applies the algebra h, and merges the effects of that:

```
{-# TERMINATING #-}
cataM f h (In xs) = join (h <$> (distr f (fmap f id (cataM f h) xs)))
```

Note that the catamorphism is bottom up: the effects from children are incurred before those of the parent. This is why we defined the primitive packers to prepend bits instead of appending them: the encoding of the root of the data structure will end up at the start of the output list, conveniently for unpacking.

The last ingredient is the distributive law. This is in essence another polytypic program, with mutually recursive definitions for functors and bifunctors (types are not needed):

```
distrF : (f : Functor)   → ⟦ f ⟧F (P a)       → P (⟦ f ⟧F a)
distrB : (f : Bifunctor) → ⟦ f ⟧B (P a) (P b) → P (⟦ f ⟧B a b)

distrF (Fix f) (In xs)   = In <$> (distrB f (fmap f id (distrF (Fix f) ) xs))

distrB (f * g) (xs , ys) = liftM2 _,_ (distrB f xs) (distrB g ys)
distrB (f + g) (inj₁ x)  = inj₁ <$> distrB f x
distrB (f + g) (inj₂ y)  = inj₂ <$> distrB g y
distrB (Const t) x       = return x
distrB (d • g) xs        = distrF d (pmap d (distrB g) xs)
distrB Par x             = x
distrB Rec x             = x
```

The variant we actually use is for bifunctors, but with pure values in the parameter positions, so we must first inject these into the monad:

```
distr f = distrB f ∘ fmap f return id
```

For example, with bit width set to 4 for brevity, the expression

```
packF ListC (packT NatTy) (toMyList (1 :: 2 :: 3 :: [])) []
```

reduces to the expression in Figure 1.

```
just (tt ,
  true ::
  false :: false :: false :: true :: -- toBits 1
  true ::
  false :: false :: true :: false :: -- toBits 2
  true ::
  false :: false :: true :: true :: -- toBits 3
  false :: [])
```

Fig. 1. The list of bits resulting from packing 1, 2, 3 is true to indicate a cons cell, then four bits representing the number 1, then similarly for the next two elements, then false to indicate a nil cell.

5.2 Unpacking as a monadic anamorphism

Let us now turn to unpacking. Being the inverse of packing, it will use the dual pattern: a *monadic anamorphism*, which is again like the ordinary anamorphism only where the coalgebra and the anamorphism itself are Kleisli arrows:

```
mutual
  {-# TERMINATING #-}
  anaM : (f : Bifunctor) → (b → P (⟦ f ⟧B a b)) → b → P (⟦ Fix f ⟧F a)
```

Unpacking is again defined in terms of three mutually recursive functions, unpackers respectively for types, functors, and bifunctors, each with the corresponding number of element unpackers as arguments:

```
  unpackT : (t : Type)       →                Unpacker ⟦ t ⟧T
  unpackF : (d : Functor)    →
            Unpacker a →                      Unpacker (⟦ d ⟧F a)
  unpackB : (f : Bifunctor)  →
            Unpacker a → Unpacker b → Unpacker (⟦ f ⟧B a b)
```

For the base types we defer to the earlier primitives:

```
  unpackT NatTy    = unpackNat
  unpackT BoolTy   = unpackBool
  unpackT UnitTy   = unpackUnit
```

For the sole functor code, we use the anamorphism:

$$\mathsf{unpackF}\ (\mathsf{Fix}\ f)\ u = \mathsf{anaM}\ f\ (\lambda\ \_\ \rightarrow \mathsf{unpackB}\ f\ u\ \mathsf{unpackUnit})\ \_$$

Note that the 'seed' of the anamorphism is the unit type: all the information driving the computation comes from the list of booleans, encoded in the monad. So the bound variable of the lambda is irrelevant, and the initial seed of the anamorphism can be inferred. As with packing, the anamorphism handles the recursive calls, so at the top level we need do nothing for the recursive positions (unpackUnit, another no-op).

And finally, there is one fairly simple case per bifunctor:

```
unpackB (f * g) u v     = liftM2 _,_ (unpackB f u v) (unpackB g u v)
unpackB (f + g) u v     = do b ← unpackBool
                             caseBit b
                               (inj₁ <$> unpackB f u v)
                               (inj₂ <$> unpackB g u v)
unpackB (Const x) u v = unpackT x
unpackB (d • g) u v     = unpackF d (unpackB g u v)
unpackB Par u v         = u
unpackB Rec u v         = v
```

For products, we unpack the left then the right components; for sums, we consume one discriminator bit in order to decide which branch to take.

Now back to the monadic anamorphism. This applies the coalgebra to the seed, makes recursive calls to generate each of the children, collects all their effects, merges the effects from the coalgebra and the recursive calls, then wraps the result up in the constructor:

```
anaM f h y = In <$> join (distr f <$> (fmap f id (anaM f h) <$> h y))
```

To illustrate the round trip, it should be the case that whatever value we take, if we pack it in front of any bit sequence then unpack the resulting sequence, that composition should succeed, and should return the original value and sequence:

```
packUnpack : { a : Set } → Packer a → Unpacker a → a → List Bool → Set
packUnpack p u x bs = (p x » u) bs ≡ just (x , bs)
```

(recall that a packer returns unit, which we then discard by »). Then we can instantiate this scheme for the types, functors, and bifunctors in our universe:

```
packUnpackT : (a : Type) → ⟦ a ⟧T →                    List Bool → Set
packUnpackT a = packUnpack (packT a) (unpackT a)

packUnpackF : (d : Functor) → (a : Type) → ⟦ d ⟧F ⟦ a ⟧T → List Bool → Set
packUnpackF d a = packUnpack (packF   d (packT a)    )
                             (unpackF d (unpackT a) )
```

```
packUnpackB : (f : Bifunctor) → (a b : Type) →
               ⟦ f ⟧B ⟦ a ⟧T ⟦ b ⟧T →                              List Bool → Set
packUnpackB f a b = packUnpack (packB    f (packT a)    (packT b)    )
                               (unpackB f (unpackT a) (unpackT b) )
```

For example, we can check the round trip property on a list of three naturals:

```
packUnpackList : ∀ (bs : List Bool) →
                   packUnpackF ListC NatTy (toMyList (1 :: 2 :: 3 :: [])) bs
packUnpackList bs = refl
```

The value we give to packUnpackList is simply refl, which indicates that (and is only type correct when) the two sides of the equivalence are definitionally equal.

6 Discussion

The technique we have used of identifying an algebraic datatype of *codes* for types drawn from some *universe* is a standard pattern in dependently typed programming. It is an instance in microcosm of the "formulation á la Tarski" that Martin-Löf [10] used in macrocosm to construct a universe of discourse for intuitionistic type theory. Another way of looking at it is specifying an embedded domain specific language for types (namely the codes), and semantics by way of a shallow embedding into the host language (namely the interpretation) [2].

This paper is literate Agda, although some of the gory details have been elided for presentation purposes. The full story can be seen in the source, which is available on GitHub [4], and typechecks at least with version 2.7.0.1 of Agda and version 2.2 of the Agda standard library.

We have seen that the polytypic programming features that Johan pioneered with PolyP can be done nowadays as 'mere programming', given a sufficiently rich language—in particular, a dependently typed one. We have chosen Agda, but Idris would work just as well.

Even Haskell is almost powerful enough these days: much of the PolyP functionality was achieved in Haskell already in 2003 [14]. But dependent types have the additional advantage that proofs become part of the language. We have exploited this briefly above: the code contains some unit tests, which are run as part of typechecking. And indeed we have exploited these tests while writing the programs: although many silly errors are ruled out by the types, it is in particular still possible to write out the wrong bit sequences.

But a powerful and informative type system like Agda's is not just there to prevent accidents. It is also hugely valuable when it comes to writing the programs in the first place: the type specifies much about the program, so with suitable interaction between the type checker and the editor—for example, in the Agda mode for Emacs—much of the program can be written automatically. Some values can be inferred; case analyses can be automatically generated; programs can be typechecked while still containing holes, and these holes can be explored with information about the goal type and the variables in scope.

Programming in this type-driven style in many ways feels like a text-based adventure game. You find yourself in a hole, with various objects at your disposal, and you have to find a way out. You keep getting sent on side-quests. Sometimes it feels like you are fighting the typechecker; but sometimes it feels like the universe is on your side, and the obstacles are magically eliminated. In recent years, Johan's research interests have shifted from programming languages to technology for education, including 'serious games': perhaps Johan can see scope for closing the circle by bringing the two back together?

References

1. Garcia, R., Järvi, J., Lumsdaine, A., Siek, J.G., Willcock, J.: An extended comparative study of language support for generic programming. Journal of Functional Programming **17**(2), 145–205 (2007). https://doi.org/10.1017/S0956796806006198
2. Gibbons, J., Wu, N.: Folding domain-specific languages: Deep and shallow embeddings. In: International Conference on Functional Programming (Sep 2014). https://doi.org/10.1145/2628136.2628138
3. Jansson, P.: PolyP source code. https://github.com/patrikja/PolyP
4. Jansson, P., Gibbons, J.: Source code for the "PolyP 30" paper. https://github.com/DSLsofMath/PolyP30
5. Jansson, P., Jeuring, J.: PolyP – A polytypic programming language extension. In: Principles of Programming Languages (POPL). pp. 470–482. ACM Press (1997). https://doi.org/10.1145/263699.263763
6. Jansson, P., Jeuring, J.: Polytypic data conversion programs. Science of Computer Programming **43**, 35–75 (2002). https://doi.org/10.1016/S0167-6423(01)00020-X
7. Jeuring, J.: Polytypic pattern matching. In: Functional Programming Languages and Computer Architecture. pp. 238–248. ACM (1995). https://doi.org/10.1145/224164.224212
8. Jeuring, J.: Generic Haskell: A language for generic programming. NWO grant 612.069.000 (2000), https://www.nwo.nl/en/projects/612069000
9. Jeuring, J., Jansson, P.: Polytypic programming. In: Launchbury, J., Meijer, E., Sheard, T. (eds.) Advanced Functional Programming. Lecture Notes in Computer Science, vol. 1129, pp. 68–114. Springer-Verlag (1996). https://doi.org/10.1007/3-540-61628-4_3
10. Martin-Löf, P.: Intuitionistic Type Theory. Studies in Proof Theory, Bibliopolis (1984), notes by Giovanni Sambin of a series of lectures given in Padua, June 1980. ISBN 88-7088-105-9.
11. Meertens, L.: Calculate polytypically! In: Kuchen, H., Swierstra, S.D. (eds.) Programming Languages: Implementations, Logics, and Programs. Lecture Notes in Computer Science, vol. 1140, pp. 1–16. Springer-Verlag (1996). https://doi.org/10.1007/3-540-61756-6_73
12. Meijer, E., Jeuring, J.: Merging monads and folds for functional programming. In: Jeuring, J., Meijer, E. (eds.) Advanced Functional Programming. Lecture Notes in Computer Science, vol. 925, pp. 228–266. Springer (1995). https://doi.org/10.1007/3-540-59451-5_7

13. Norell, U.: Towards a Practical Programming Language Based on Dependent Type Theory, vol. 32. Chalmers University of Technology (2007), https://www.cse.chalmers.se/~ulfn/papers/thesis.pdf, ISBN 978-91-7291-996-9.
14. Norell, U., Jansson, P.: Polytypic programming in Haskell. In: Trinder, P., Michaelson, G.J., Peña, R. (eds.) Implementation of Functional Languages. Lecture Notes in Computer Science, vol. 3145, pp. 168–184. Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/978-3-540-27861-0_11

Thank you Johan!

José Pedro Magalhães

Two people fundamentally shaped my academic (and then professional) career: José Nuno Oliveira by showing me the beauty and power of Haskell in 2003, and Johan Jeuring by introducing me to the amazing world of generic programming in 2004. It was then that I started considering the idea of doing a PhD, and indeed three years later, with a lot of assistance from Johan, my PhD grant proposal was successful. I started my PhD in generic programming in February 2008 under Johan's supervision at Utrecht University.

To be truly honest, it was not only the topic of generic programming that attracted me to a PhD with Johan. It was also Johan's teaching style: methodical, measured, and always calm. He maintained this demeanour throughout my PhD; I can't easily recall any moment when I thought Johan was visibly angry or impatient, and he always seemed to be in full control of the situation, no matter how many deadlines approached. I tried to carry this aspect of Johan through my own career, and perhaps this influenced me even more than any deep theoretical programming language insight.

I should also say, for avoidance of doubt, that Johan was an excellent academic PhD supervisor. As I wrote in my thesis acknowledgements back in 2012:

> Johan was always available to correct my mistakes, provide me with new insights and research directions, look over my code, or just discuss some random ideas. I could not have asked more from a supervisor, and the success of my PhD is in great part due to him.

So thank you, once again, Johan, for believing in my potential, and for all the guidance through the years. I can confidently say I would not be where I am today without your influence.

Incremental Optimal-Fit Line Breaking

Lambert Meertens

CWI*

Abstract. The problem of breaking paragraphs into lines can be formulated as an optimization problem. Dynamic programming is used for the efficient solution of a general version. At a meeting of IFIP Working Group 2.1 in 1995 I presented a derivation of an $O(1)$ algorithm for the incremental version of the general problem, but it was not published. This paper describes the approach.

0 Historical remarks

A standard task in typesetting is breaking paragraphs into lines. Early typesetting algorithms, such as that of Unix's *roff* (1970), use a greedy "first fit" method: put as much on the first line as will fit, but no more, and keep using the same method for assigning the remaining text (if any) to subsequent lines. This can result in fits that are much worse than the best fit under any reasonable cost function. (See Figure 1.)

Knuth & Plass [1] were the first to present a dynamic-programming algorithm for finding the optimal fit, making this a tractable problem with an $O(n)$ run-time cost. Richard Bird [2] gave a derivation of this algorithm, in the squiggol style, for a simplified version of the problem that ignores the possibilities of proportional spacing and hyphenation. Next, Oege de Moor [3] gave a derivation for the general problem.

While optimal, these algorithms were not incremental. In a WYSIWYG system showing a user the optimally formatted result of a paragraph in real time, updating the layout as the user edits the input text, a non-incremental algorithm has to be re-applied to the full text of the paragraph at every keystroke. For long paragraphs, this easily leads to unacceptable update delays.

Johan Jeuring [4] was the first to give an incremental algorithm for finding the optimal fit. Like Bird's presentation, his was a derivation in the squiggol style, and

* Although I retired from CWI in 1998, the work reported on here was done while I was at CWI.

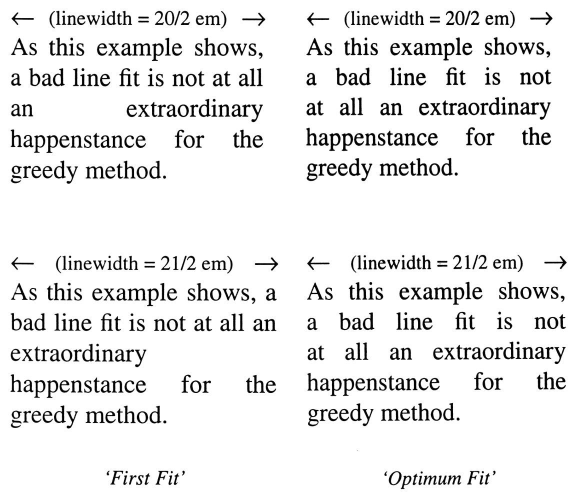


Fig. 1. 'First Fit' and 'Optimal Fit' on the same input, side by side.

also like Bird's, the derivation was for a simplified version of the problem, making it unsuitable for practical deployment.

At Meeting 46 of IFIP Working Group 2.1, held in Renkum, the Netherlands, in January 1995, I presented a derivation of an $O(1)$ incremental algorithm for the general problem. Until now, it has not been published (or otherwise been publicly available). So now here it is. The presentation here mostly follows that of the original, handwritten handout [5]. The latter freely applied handwaving, and so does the presentation here.

1 Notation

Section notation. Given an infix operator $\odot : A \times B \to C$, we define, for $a \in A$ and $b \in B$, functions $(a\,\odot) : B \to C$ and $(\odot\, b) : A \to C$ by:

$$(a\,\odot)(x) = a \odot x\ ;$$

$$(\odot\, b)(x) = x \odot b.$$

Lists. We will be working with *lists*, that is, finite homogeneous sequences (having all its elements from the same domain). The set of lists whose elements are members of a given set A will be denoted by A^*. The set $A^+ \subset A^*$ consists of the nonempty lists over A.

The length (number of elements) of list x is given by $\#x$. The empty list will be written as $[\,]$, the singleton list containing a single element a will be written as $[a]$, that of two elements a and b as $[a, b]$, and so on. The concatenation of two lists x and y is denoted by $x +\!\!+ y$. Prepending an element a to a list x will be denoted by the operator $\rightarrowtail$, also known as *cons*, while appending it will be denoted by the operator $\leftarrowtail$, also known as *snoc*. They can be defined more formally by:

$$a \rightarrowtail x \;=\; [a] +\!\!+\, x\;;$$

$$x \leftarrowtail a \;=\; x +\!\!+ [a].$$

The function $([\,]\leftarrowtail) : A \to A^+$ forms a singleton list. We could equally have written $(\rightarrowtail[\,])$, but will use the symmetric notation $[\cdot]$, so $[\cdot](a) = [a]$.

Map. Given function $f : A \to B$, its extension to the map function $f^* : A^* \to B^*$ is characterized by:

$$\begin{aligned} f^*\,[\,] &= [\,]; \\ f^*\,[a] &= [f(a)]; \\ f^*(x +\!\!+ y) &= f^*(x) +\!\!+ f^*(y). \end{aligned}$$

For example, if *succ* is the function assigning to each letter in the Latin alphabet its successor (with $succ(\mathtt{Z}) = \mathtt{A}$ to complete the circle), we have $succ^*[\mathtt{H}, \mathtt{A}, \mathtt{L}] = [succ(\mathtt{H}), succ(\mathtt{A}), succ(\mathtt{L})] = [\mathtt{I}, \mathtt{B}, \mathtt{M}]$.

Map distributes over function composition, as expressed by the functional identity

$$(f \circ g)^* = (f^*) \circ (g^*).$$

Reduce. Let $(\mathcal{A}, \odot, \nu_\odot)$ be a monoid with carrier set $\mathcal{A}$ and a binary associative operation $\odot : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ with neutral element $\nu_\odot$. Then $\odot/ : \mathcal{A}^* \to \mathcal{A}$ is characterized by:

$$\begin{aligned} \odot/[\,] &= \nu_\odot; \\ \odot/[a] &= a; \\ \odot/(x +\!\!+ y) &= (\odot/x) \odot (\odot/y). \end{aligned}$$

For example, $+/[4, 6, 3] = 4 + 6 + 3 = 13$, and $+\!\!+/[[a], [\,], [b, c]] = [a] +\!\!+ [\,] +\!\!+ [b, c] = [a, b, c]$.

Filter. Let $\top$ and $\bot$ stand for the truth values *true* and *false*, together forming $\mathbb{B} = \{\top, \bot\}$. Given a predicate $p : A \to \mathbb{B}$ defined on list elements that tests whether they are "worthy" of being kept, we can combine reduce and map to define a function $p^{\triangleleft} : A^* \to A^*$ that filters out the worthy elements of a list and discards the rest. We first define a (purely auxiliary) function $p^? : A \to A^*$ defined by:

$$p^?(a) = \begin{cases} [a] \text{ if } p(a) = \top \\ [\,] \text{ if } p(a) = \bot \end{cases}.$$

Our filter function $p^{\triangleleft}$ is then defined as:

$$p^{\triangleleft}(x) = +\!\!+/\,(p^{?})^{*}(x).$$

For example, if *odd* tests whether a natural number is odd, we have

$$\begin{aligned} odd^{\triangleleft}\,[3,4,5] &= +\!\!+/\,(odd^{?})^{*}\,[3,4,5] \\ &= +\!\!+/\,[odd^{?}(3), odd^{?}(4), odd^{?}(5)] \\ &= +\!\!+/\,[[3],[\,],[5]] = [3]+\!\!+[\,]+\!\!+[5] = [3,5]. \end{aligned}$$

Application to sets. Finite sets can be represented by lists; for example, $[2,3,1]$, and $[3,1,2,1]$ are different representations of the set $\{1,2,3\}$. If we think of concatenation $+\!\!+$ as implementing set union $\cup$ for sets represented by lists, the definition of f^{*} given above works equally well for sets represented by lists, regardless of the particular representation. This holds likewise for $+\!\!+/$ (implementing $\bigcup$) and $p^{\triangleleft}$. For this to work for $\odot/$ in general, the operation $\odot$ needs to be also commutative and idempotent. Since $\wedge$ is both, $\wedge/$ can be thought of as working on sets of truth values.

Inits, tails and segments. Function $inits : A^{*} \to A^{**}$ returns a list of all initial segments of its argument, so $inits\,[a,b,c] = [[\,],[a],[a,b],[a,b,c]]$. It can be defined by

$$\begin{aligned} inits\,[\,] &= [[\,]]; \\ inits(a \succ\!\!+ x) &= [[\,]] +\!\!+ (a \succ\!\!+)^{*}(inits(x)). \end{aligned}$$

Its mirror image, function $tails : A^{*} \to A^{**}$, returns a list of all tail segments of its argument, so $tails\,[a,b,c] = [[a,b,c],[b,c],[c],[\,]]$. It can be defined by

$$\begin{aligned} tails\,[\,] &= [[\,]]; \\ tails(x +\!\!\prec a) &= (+\!\!\prec a)^{*}(tails(x)) +\!\!+ [[\,]]. \end{aligned}$$

All segments of a list are then given by:

$$segs = +\!\!+/ \circ tails^{*} \circ inits.$$

So $segs\,[a,b,c] = [[\,],[a],[\,],[a,b],[b],[\,],[a,b,c],[b,c],[c],[\,]]$. A list of length n has in total $(n{+}1)(n{+}2)/2$ segments, of which $n{+}1$ are empty.

Partitions. A partition of a list x is a list whose elements are nonempty segments of x, such that, concatenated together, they give us back the original list x. For example, $[[7,0],[7],[2,4,5]]$ is a partition of $[7,0,7,2,4,5]$. Defining auxiliary predicate *pop* (for "populated") on lists by $pop(x) = (\#x > 0)$, function $parts : A^{*} \to (A^{+})^{*}$ is such that $parts(x)$ represents the set of partitions

$$\{xs \mid xs = pop^{\triangleleft}(xs) \;\wedge\; (+\!\!+/\,xs = x)\}.$$

Optimum selection w.r.t a criterion. Let $f : A \to \mathbb{P}$ be a function from some domain A to a totally ordered codomain $\mathbb{P}$, representing a criterion that will guide the selection of an optimal choice. Rather than awarding points for goodness of fit, we measure the *cost* of a candidate, so a lower score means a better fit. (The letter $\mathbb{P}$ is the first letter of *penalty*.) For our application we can use the extended set of nonnegative reals $\{x \in \mathbb{R} \mid 0 \leq x\} \cup \{+\infty\}$, while using $\mathbb{N}$ or $\mathbb{Q}$ instead of $\mathbb{R}$ also works. The value $+\infty$ is meant to be given to a choice that is so egregiously bad that it can be discarded *prima facie* (possibly implemented as a number that is so high, such as 999999, that all other choices have a lower penalty).

We define:

$$x \downarrow_f y = \begin{cases} x \text{ if } f(x) \leq f(y) \\ y \text{ if } f(y) < f(x) \end{cases} .$$

A determinate choice between two equally good candidates — with this definition the first of the two — is pragmatically desirable.

2 Problem statement

We now have the ingredients to specify the problem more formally.

The input text is not just a list of characters. Instead, it is a preprocessed list of chunks determined by the points in the raw input text where a linebreak might occur, which can be in the middle of a word if hyphenation is possible. The begin point and end point of the text are also counted as break points, possibly the only break point if the text is empty. These (potential) break points are used to break up the text in chunks, and the list of these chunks forms the input. For example, the raw input

an incredibly important development

can be chunked (for a system allowing hyphenation) into this list of 12 chunks:

$[\mathsf{an}^{\sqcup}, \mathsf{in}^{-}, \mathsf{cred}^{-}, \mathsf{i}^{-}, \mathsf{bly}^{\sqcup}, \mathsf{im}^{-}, \mathsf{por}^{-}, \mathsf{tant}^{\sqcup}, \mathsf{de}^{-}, \mathsf{vel}^{-}, \mathsf{op}^{-}, \mathsf{ment}^{\sqcup}]$,

in which marking a chunk with $^{\sqcup}$ signals that in typesetting the chunk should be followed by a blank space unless it is the last chunk on a line, while $^{-}$ signals that the chunk should be followed by a hyphen if at the end of a line, and otherwise be joined without intervening space to the next chunk.

The treatment in this article will not consider how these chunk lists are represented, but one possibility for a practical implementation is that of a doubly-linked list, so that edit operations in the middle of the text need no more than $O(1)$ cost.

Given the input list $x : A^{+}$ of text chunks, the partitions of the list correspond to the potential ways of breaking up the text into lines. We assume that a function $\mathit{slpen} : A \to \mathbb{P}$ is given for assigning a penalty to a single line of a partition. The total penalty $pen = +/ \circ slpen^*$, obtained by summing the individual penalties of the lines of a partition, can be used to determine an optimal-fit line breaking of a given paragraph; in a formula,

$$\downarrow_{pen}/\ parts(x).$$

Function *pen* is monotonic with respect to concatenation:

$$pen(x) \leq pen(y) \ \Rightarrow\ pen(u +\!\!\!+ x +\!\!\!+ v) \leq pen(u +\!\!\!+ y +\!\!\!+ v)$$

for all lists x, y, u and v. An equivalent formulation of this monotonicity property is the statement that concatenation $+\!\!\!+$ distributes over optimal choice $\downarrow_{pen}$:

$$u +\!\!\!+ (x \downarrow_{pen} y) +\!\!\!+ v = (u +\!\!\!+ x +\!\!\!+ v) \downarrow_{pen} (u +\!\!\!+ y +\!\!\!+ v).$$

If there is no way a chunk segment s can fit on a single line, partitions having s as an element should be avoided. This is modeled by then taking $slpen(s) = +\infty$. Other than that, the single-line penalty function can be almost arbitrary; we shall introduce a pragmatic assumption below.

Generality of the penalty function is important for quality typesetting. A simple penalty function for a single line found in the literature is the combined width of blank space, sometimes referred to as *waste*. However, this is overly simplistic, particularly for justified typesetting, in which both margins are adjusted. Then the combined blank space is divided between the words on the line. The total width of blank space, divided evenly into the three spaces between four words, may look just fine. When the same amount of space separates just two words, the result may be jarring. Also, sometimes a better overall fit can be achieved by squeezing the words on a line just a tiny bit together. This should come at a cost; the squeezing should incur a penalty, even if the “waste” is reduced.

Another issue is that not all break points are created equal. All else being the same, breaking between two words is better than resorting to hyphenation to break inside a word, and breaking after a full sentence or a comma is better than breaking just anywhere between two words. In the famous sentence `ALL YOUR BASE ARE BELONG TO US`, breaking the line as in

`ALL YOUR BASE / ARE BELONG TO US`

is easier on the reader than

`ALL YOUR BASE ARE / BELONG TO US.`

A limitation of the specification is that all lines are treated the same, so it does not accommodate a varying distance between the margins, as when text flows around a figure. Pagination also falls outside the scope of the problem as considered here. The last line of a paragraph should also get a special treatment; the line is not adjusted, and an ample amount of blank space should not be penalized. Fortunately, this can easily be accommodated. The invoker *slpen** of the application of *slpen* to each of the chunk segments of a partition "knows" when this function is applied to the last segment and can then substitute a last-line variant.

Pragmatic assumption. Since lines are not infinitely wide, and chunks are not infinitely thin, there is an upper bound on the number of chunks that can fit on any line. We assume that a reasonably low value β of such an upper bound is known, and that chunk segments that are too large to fit on a line get the maximal penalty, so that

$$\#s > \beta \;\Rightarrow\; \mathit{slpen}(s) = +\infty.$$

3 Theory

Dags. A *dag* (directed acyclic graph) $(\mathcal{N}, \mathcal{E})$ consists of

- a set of *nodes* $\mathcal{N}$ and
- a set of *edges* $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$,
- such that the transitive closure $\mathcal{E}^+$ is irreflexive — there is no path from a node to itself.

The meaning of $(s, t) \in \mathcal{E}$ is that there is an edge *from* node s *to* node t.

Partial orders. A *partial order* $(\mathcal{A}, \sqsubseteq)$ consists of

- a set $\mathcal{A}$ and
- a relation $\sqsubseteq\ \subseteq\ \mathcal{A} \times \mathcal{A}$,
- such that $\sqsubseteq$ is transitive, reflexive and antisymmetric.

Po-dags and weighted dags. A partial order $(\mathcal{A}, \sqsubseteq)$ can be turned into a dag as follows. Take the irreflexive relation $(\sqsubset) = (\sqsubseteq) \setminus I_{\mathcal{A}}$, where $I_{\mathcal{A}} = \{(a, a) \mid a \in \mathcal{A}\} \subseteq \mathcal{A} \times \mathcal{A}$. Then $(\mathcal{A}, \sqsubset)$ is a dag, which we refer to as the *po-dag* of the partial order. The partial order can be recovered from its po-dag by using $(\sqsubseteq) = (\sqsubset) \cup I_{\mathcal{A}}$.

Given a list x of length $\#x = n$, there is a bijection between the elements of $\mathit{parts}(x)$ and the paths in the po-dag of $(\{0, ..., n\}, \leq)$. Each edge (s, t) in a path corresponds to a segment of length $t - s$ of the partition. This bijection plays an important role in what follows. If the edges have "lengths", and the length of an edge is the penalty of the corresponding spanned segment, a shortest path from 0 to n in the po-dag corresponds to an optimal-fit line breaking of paragraph x. Such valuations of edges are more commonly called *weights*.

A $\mathcal{V}$-weighted dag $(\mathcal{N}, \mathcal{E}, \mathcal{V}, w)$ comprises:

- a dag $(\mathcal{N}, \mathcal{E})$,
- a set of weights $\mathcal{V}$, and
- a weight function $w : \mathcal{E} \to \mathcal{V}$.

In our application, $\mathcal{V} = \mathbb{P}$. We will use matrices, treated later, to represent weighted dags.

Dioids. A *dioid* (a.k.a. *semiring*) $(\mathcal{A}, \oplus, \nu_\oplus, \otimes, \nu_\otimes)$ comprises:

- a commutative monoid $(\mathcal{A}, \oplus, \nu_\oplus)$ and
- a monoid $(\mathcal{A}, \otimes, \nu_\otimes)$ over the same carrier,
- such that $\otimes$ distributes over $\oplus$.

The distributivity property implies (and is implied by) the functional identities

$$(a\otimes) \circ \oplus/ = \oplus/ \circ (a\otimes)^*;$$
$$(\otimes\, a) \circ \oplus/ = \oplus/ \circ (\otimes\, a)^*.$$

It also implies that $\nu_\oplus$ is a zero of operation $\otimes$.

Dioids include:

- rings, e.g. $(\mathbb{Z}, +, 0, \times, 1)$,
- complete semilattices, e.g. $(\mathbb{P}, \downarrow_{\mathrm{id}}, +\infty, +, 0)$, also called a "tropical semiring",
- regular algebras, introduced later on.

Sparse and full matrices. Let $\mathbb{N}_n$ stand for the set $\{0, ..., n\}$ (of size $n{+}1$). A (sparse) $\mathcal{V}$-matrix $(\mathcal{E}, \mathcal{V}, M)$ consists of

- a set of index pairs $\mathcal{E} \subseteq \mathbb{N}_n \times \mathbb{N}_n$ for some $n \in \mathbb{N}$,
- a set of values $\mathcal{V}$,
- a function $M : \mathcal{E} \to \mathcal{V}$.

Instead of $M(i, j)$ we use the conventional notation $M_{i,j}$, with the index pair as a subscript.

A function $f : \mathcal{V} \to \mathcal{W}$ can be extended to a map function f^* from $\mathcal{V}$-matrices to $\mathcal{W}$-matrices with the same set of index pairs by defining

$$(f^*(M))_{i,j} = f(M_{i,j}),$$

which amounts to replacing $M : \mathcal{E} \to \mathcal{V}$ by $f \circ M : \mathcal{E} \to \mathcal{W}$.

When $\mathcal{E} = \mathbb{N}_n \times \mathbb{N}_n$, we have a full square matrix. By adjoining a new element $\bot$ (representing a "missing entry") to $\mathcal{V}$, defining $\mathcal{V}_\bot = \mathcal{V} \cup \{\bot\}$, we can conceptually extend a sparse $\mathcal{V}$-matrix $(\mathcal{E}, \mathcal{V}, M)$ with $\mathcal{E} \subset \mathbb{N}_n \times \mathbb{N}_n$ to a full $\mathcal{V}_\bot$-matrix $(\mathbb{N}_n \times \mathbb{N}_n, \mathcal{V}_\bot, M_\bot,)$ where

$$(M_\bot)_{i,j} = \begin{cases} M_{i,j} & \text{if } (i,j) \in \mathcal{E} \\ \bot & \text{if } (i,j) \notin \mathcal{E} \end{cases}.$$

The sparse matrix can be reconstructed from the full version by taking $\mathcal{E} = \{(i, j) \mid (M_\bot)_{i,j} \neq \bot\}$. In what follows, we shall switch tacitly between the sparse and the full view.

As is somewhat obvious from their formal definitions, $\mathcal{V}$-weighted dags whose nodes are natural numbers are also $\mathcal{V}$-matrices. The converse is not necessarily true; a $\mathcal{V}$-matrix viewed as a graph is not necessarily acyclic. But one is a subspecies of the same species of animal in the mathematical zoo, so rather than one being "represented" by the other, they actually represent different modes of contemplating the same object. Ultimately, the partitions of a chunked input text, together with a penalty function, can be viewed as a $\mathbb{P}$-weighted dag and therefore as a $\mathbb{P}$-matrix. A matrix thus obtained is an upper triangular matrix.

Nested-list view. We can encode a full matrix $(\mathbb{N}_n \times \mathbb{N}_n, \mathcal{V}_\perp, M_\perp)$ as a vector of column vectors. Since these vectors have finite lengths, they can be represented as lists. The full matrix view can be recovered from the nested-list view in an obvious way.

We shall also switch tacitly between the structure view and the nested-list view on matrices. For example, if the full version of M is an $n \times n$ matrix, and v is a vector of length $n{+}1$, the full version of $M \mathbin{+\!\!<} v$ is an $(n{+}1) \times (n{+}1)$ matrix. Snoccing, successively, vectors of length 1, 2, ... onto an initially empty matrix results in an upper triangular matrix.

Matrix dioids. A dioid $(\mathcal{A}, \oplus, \nu_\oplus, \otimes, \nu_\otimes)$ induces a dioid on A-matrices sharing their (full) index-pair set $\mathbb{N}_n \times \mathbb{N}_n$, whose "funny plus" and "funny times" are defined by

$$\begin{aligned} (M \oplus N)_{i,j} &= M_{i,j} \oplus N_{i,j}; \\ (M \otimes N)_{i,j} &= \textstyle\bigoplus_k \, (M_{i,k} \otimes N_{k,j}). \end{aligned}$$

As with conventional matrices from linear algebra, these operations preserve upper-triangularity.

Regular algebras. A *regular algebra* $(\mathcal{A}, \sqsubseteq, \circ, I)$ comprises:

- a complete lattice $(\mathcal{A}, \sqsubseteq)$, for which we denote the join by $\sqcup$ and bottom by $\perp$, and
- a monoid $(\mathcal{A}, \circ, I)$,
- such that $(x \circ)$ and $(\circ\, x)$ are universally $\sqcup$-junctive for all $x \in \mathcal{A}$ — that is, they are monotonic and continuous.

Given a regular algebra $(\mathcal{A}, \sqsubseteq, \circ, I)$, we have that $(\mathcal{A}, \sqcup, \perp, \circ, I)$ is a dioid.

If we apply the matrix dioid construction to a dioid derived from a regular algebra, the resulting matrix dioid also corresponds to a regular algebra, but now a regular algebra of matrices.

Examples of regular algebras:

- $(\mathbb{P}, \leq, +, 0)$.

- For any monoid $(\mathcal{A}, \oplus, \nu_\oplus)$ we have that $(\mathcal{P}(\mathcal{A}), \subset, \mathsf{X}_\oplus, \{\nu_\oplus\})$ is a regular algebra, in which $\mathcal{P}(\mathcal{A})$ stands for the powerset of $\mathcal{A}$, and the operator $\mathsf{X}_\oplus$ is defined by
  $x \mathbin{\mathsf{X}_\oplus} y = \{a \oplus b \mid a \in x \wedge b \in y\}.$

For the latter construction, it is worth remarking that the monoid $(\mathcal{A}^*, +\!\!+, [\,])$ gives rise to the well-known *language algebra*, the regular algebra of languages over an alphabet $\mathcal{A}$.

In a regular algebra we can define the *Kleene plus* operation $\_\_^+ : \mathcal{A} \to \mathcal{A}$, informally described by $x^+ = x \sqcup x \circ x \sqcup x \circ x \circ x \sqcup \ldots$. A precise characterization is given by

$$c = x^+ \;\Leftrightarrow\; c \circ c = c \;\wedge\; \forall(y : y \circ y \sqsubseteq y : x \sqsubseteq y \Leftrightarrow c \sqsubseteq y).$$

For the matrix regular algebra, the Kleene plus is

$$M^+ = M \;\oplus\; M^2 \;\oplus\; M^3 \;\oplus\; \ldots,$$

in which $M^1 = M$ and $M^{n+1} = M^n \otimes M$. If M is a triangular matrix with index-pair set $\mathbb{N}_n \times \mathbb{N}_n$, all powers M^k, for $k > n$, are the same; they are all equal to the neutral element of the matrix operation $\oplus$, the matrix all of whose entries are the neutral $\nu_\oplus$ of the carrier. It follows that the computation can be performed in a finite number of steps.

If M is triangular, then, by the preservation of triangularity by the operations $\oplus$ and $\otimes$, so is M^+. It follows that M^+ can be computed incrementally, column-wise, by the repeated application of the identity

$$(M \mathbin{+\!\!\!\ltimes} v)^+ = M^+ \mathbin{+\!\!\!\ltimes} (M^+ \otimes v).$$

Dioid morphisms and regular-algebra morphisms. Both types of morphisms are defined in the obvious way. So a dioid morphism h satisfies $h(x \oplus y) = h(x) \oplus' h(y)$, and so on, where the prime indicates this is the operation of the target dioid. Note that a regular-algebra morphism satisfies, in particular, $h \circ \bigsqcup = \bigsqcup' \circ h^*$.

If $(\mathcal{A}, \oplus, \nu_\oplus, \otimes, \nu_\otimes)$ is a dioid, the function $\oplus/ : \mathcal{P}(\mathcal{A}) \to A$ is a dioid morphism, from dioid $(\mathcal{P}(\mathcal{A}), \cup, \{\}, \mathsf{X}_\otimes, \{\nu_\otimes\})$ to dioid $(\mathcal{A}, \oplus, \nu_\oplus, \otimes, \nu_\otimes)$. If the target dioid is a regular algebra, this is also a regular-algebra morphism.

Finally, if h is a regular-algebra morphism, then so is h^* on the corresponding matrix algebra.

4 Development

The time has come to apply all this machinery to the problem. We begin by creating, seemingly, more work. The function

$$Segs : A^* \to A^{***} = tails^* \circ inits$$

turns a given list into a matrix of all segments of the given list (using the nested-list view of a matrix). We have seen this function before, anonymously, in the definition of function *segs* (with a lower-case *s*). Using lower-case *segs*, upper-case *Segs* could have been defined by:

$$segs = +\!\!+/ \circ Segs \;:\; A^* \to A^{**}.$$

The partitions function mapped over a segment matrix gives rise to an important identity:

$$parts^* \,(Segs(x)) = (\{[\cdot]\}^* \,(Segs(x)))^+,$$

in which $\{[\cdot]\}$ maps a to $\{[a]\}$ and the Kleene plus is that of the regular matrix algebra derived from the language algebra. This is a formal expression of the correspondence, mentioned earlier, between partitions and paths. We map the optimal selection over the segment matrix and calculate:

$$\begin{aligned}
& (\downarrow_{pen}/ \circ parts)^* \,(Segs(x)) \\
= \;& \\
& \downarrow_{pen}/^* \,(parts^* \,(Segs(x))) \\
= \;& \\
& \downarrow_{pen}/^* \,((\{[\cdot]\}^* \,(Segs(x)))^+) \\
= \;& \\
& ((\downarrow_{pen}/ \circ \{[\cdot]\})^* \,(Segs(x)))^+. \\
= \;& \\
& ([\cdot]^* \,(Segs(x)))^+.
\end{aligned}$$

The disappearance of $\downarrow_{pen}$ from the final expression deserves an explanation. While the Kleene plus in the earlier expressions is that of the regular matrix algebra derived from a $(\cup, \mathsf{X}_{+\!\!+})$ algebra, in the final step the morphism $\downarrow_{pen}/$ turns it into the Kleene plus of the regular matrix algebra derived from a $(\downarrow_{pen}, +\!\!+)$ algebra. So $\downarrow_{pen}$ has not truly vanished; the operation is hiding in the application of $\_\_^+$.

There is no need to compute the full matrix; if $\#x = n$,

$$\begin{aligned}
x &= \quad Segs(x)_{0,n}, \\
parts(x) &= (parts^* \,(Segs(x))_{0,n},
\end{aligned}$$

and so on. So we are actually only interested in the top row.

Letting $M_{i,*}$ stand for the i th row $[M_{i,0}, M_{i,1}, M_{i,2}, ...]$ of M, we have, for the top row of interest:

$$((M \dashv\!\!< v)^+)_{0,*} = M^+_{0,*} \dashv\!\!< (M^+_{0,*} \otimes v),$$

in which $\otimes$ is the "scalar product" of two vectors. Therefore the function

$$f = (\downarrow_{pen}/ \circ parts)^* \circ inits$$

can be computed in a scan, using

$$f(x \mathbin{+\!\!<} a) = f(x) \mathbin{+\!\!<} (f(x) \otimes tails(x \mathbin{+\!\!<} a)).$$

By symmetry, there is an analogous expression for $g(a \mathbin{>\!\!+} y)$ for the mirror-image function $g = (\downarrow_{pen}/ \circ parts)^* \circ tails$.

The stage is now set for the final act, incrementalization.

The object on which the computation proceeds is a pair of chunk lists (x, y), being the chunks *before* and *after* the edit cursor. The pair (x, y) represents the current (chunked) input text $x \mathbin{+\!\!+} y$, but with the edit focus represented. (If the cursor actually falls *inside* a chunk, we can treat this as if the cursor is *behind* that chunk.) The cursor movements left and right correspond then to

$$(x \mathbin{+\!\!<} a, y) \underset{\mathsf{RIGHT}}{\overset{\mathsf{LEFT}}{\rightleftarrows}} (x, a \mathbin{>\!\!+} y).$$

The edit operations of inserting a chunk a and deleting a chunk correspond to

$$(x, y) \underset{\mathsf{DELETE}}{\overset{\mathsf{INSERT}\ a}{\rightleftarrows}} (x \mathbin{+\!\!<} a, y).$$

Changes inside a chunk can be treated as the deletion of the chunk followed by the insertion of the changed chunk.

Incrementalization is not yet achieved by incrementally maintaining the pair of lists $(f(x), g(y))$, updating these on each operation. These lists need to be "glued" together to find $f(x \mathbin{+\!\!+} y)_n$, where $n = \#(x \mathbin{+\!\!+} y)$.

To this end, we extend $f(x)$ to $f(x \mathbin{+\!\!+} z)$, where z is a sufficiently long initial segment of y, meaning that either $pen(z) = \bot$ (no need to look further because all the rest will be $\bot$'s) or $z = y$ (we cannot go further).

The claim that we need not look further after meeting a $\bot$ is a theorem of regular algebra. Specifically, the assumption $b \circ a = \bot$ implies $b \circ a^* = b$. Using this, we also obtain that this assumption implies the equality

$$
\begin{array}{ll}
 & (a \sqcup b)^+ \\
= & \qquad \{\text{closure}\} \\
 & (a \sqcup b) \circ (a \sqcup b)^* \\
= & \qquad \{\text{regular algebra}\}
\end{array}
$$

$$
\begin{array}{ll}
 & (a \sqcup b) \circ a^* \circ (b \circ a^*)^* \\
= & \quad \{\text{assumption}\} \\
 & (a \sqcup b) \circ a^* \circ b^* \\
= & \quad \{ \circ \text{ distributes over } \sqcup \} \\
 & (a \circ a^* \circ b^*) \sqcup (b \circ a^* \circ b^*) \\
= & \quad \{\text{left: closure; right: assumption}\} \\
 & (a^+ \circ b^*) \sqcup (b \circ b^*) \\
= & \quad \{\text{closure}\} \\
 & (a^+ \circ (I \sqcup b^+)) \sqcup b^+ \\
= & \quad \{ \circ \text{ distributes over } \sqcup \} \\
 & a^+ \sqcup (a^+ \circ b^+) \sqcup b^+.
\end{array}
$$

The "gluing" can therefore be achieved by computing

$$
\begin{array}{ll}
 & f(x +\!\!+ z) \otimes g(y) \\
= & \\
 & \downarrow_{pen} / \ \{(f(x +\!\!+ z)_{(\#x)+k} +\!\!+ g(y)_k \mid 0 \leq k \leq \#z\}.
\end{array}
$$

Using the "banana split" technique of paramorphisms, we can also incrementalize the computation of the *pen* values.

The claim that the incrementalized computation takes time $O(1)$ depends crucially on the pragmatic assumption introduced early on. It means that $\#z$, in the extension of x above, is bounded by β. For each edit operation, the number of elementary operations in the RAM model needed for gluing is then bounded by a constant times β^2, which is also a constant independent of the size of the text being edited.

Earlier on, a doubly-linked list was mentioned as a practical data structure for the chunk list. By extending each of the pointers in a pair of pointers to a vector of pointers of length β, decorated with the associated *pen* values, the matrix (or path graph, if you wish) is represented implicitly, keeping the administration all with the input.

References

1. Knuth, Donald E. & Michael F. Plass: Breaking paragraphs into lines. *Software: Practice & Experience* **11**:11, 1119–1184 (1981).
2. Bird, Richard S.: Transformational programming and the paragraph problem. *Science of Computer Programming* **6**, 159–189 (1986).

3. de Moor, Oege: *Categories, Relations and Dynamic Programming.* Ph.D. Thesis, Oxford University Computing Laboratory Programming Research Group, PRG-98 (1992).
4. Jeuring, Johan: *Theories for Algorithm Calculation.* Ph.D. Thesis, Utrecht University (1993).
5. Meertens, Lambert: Incremental Optimum-Fit Line Breaking. IFIP WG2.1 working paper 720 (REN-5) (1994).

Algorithmics for Unicians

Steven Pemberton

CWI, Amsterdam
http://cwi.nl/~steven/

Abstract. There are a number of articles I could have written for this *Liber*, including "*Why I Never Write Articles Using Latex*" (I probably got the orthography wrong there, but only members of the cult know and care), but instead I present you with *The Article I Would Have Written For* The Squiggolist *If It Were Still Publishing*.
Alas I can't be at your Jubilee event: I am giving a course at Oxford University, as I do every year at this time. Have fun. Wish I could be there!

Motivation

Consider the Unix command '`wc`' (word count). Given the parameter `-l`, it counts the lines of each file of its input, and outputs the counts and a total:

```
$ wc -l *.c
 11 box.c
280 symbols.c
 88 transys.c
379 total
```

There are two remarks to be made here: firstly it's strange to be counting lines with a program for counting words; secondly the final total line, while useful in many cases, actually goes against a principal of Unix of only producing what is asked for.

If you want more, then you must pipe the output to another program. This, according to Thompson himself, is why originally `ls` didn't produce its output in columns: there are programs to do that, and single column output is much more useful for piping to other programs.

So, let us postulate, for the sake of this article, the existence of a command `lc` that counts the number of lines of each file, giving the above output without the final total line:

```
$ lc *.c
 11 box.c
280 symbols.c
 88 transys.c
```

To produce the total line, we now need a command to sum the numbers given to it in its input. We could call this command `sum`, but that name has already been bagged. We could use `sigma`, but since Unix command names don't have to be alphabetic, I'm going to use `+`:

```
$ lc *.c | +
379
```

(In fact, it's such a useful command, I'm surprised it isn't a standard command. I use it all the time, and if you want to use it, here it is:

```
# Sum a column of numbers
# Usage: + [+/-n] [-fc] [file] ...
#  +/-n: column number offset ala sort. Default: uses column 1
#  -fc: field separator character ala awk. Default: space or tab
col=1
files=
fs=
for i
do
  case $1 in
  -[0-9]*) col=`expr $1 : '-\(.*\)'`; shift;;
  +[0-9]*) col=`expr $1 : '+\(.*\)' + 1`; shift;;
  -f*) fs=`expr $1 : '-f\(.\)'`; shift;;
  -*) echo Usage: `basename $0` [+/-n] [-fchar] [file] ...; exit 1;;
  *) files="$files $1"; shift;;
  esac
done
case $fs in
"") awk "{s+=\$$col}
         END {print s}" $files ;;
*)  awk "-F$fs" "{s+=\$$col}
               END {print s}" $files ;;
esac
```

)

Now any Unix-literate computer user knows that

```
lc *.c | +
```

would produce the same[1] output as

```
cat *.c | lc
```

But why? And can we draw any general conclusions from this?

[1] Unix allows the last line of a text file not to be terminated. This article assumes all text files have a terminated last line.

Some Notation

First let us agree on some notation and terminology. Let us assume for the sake of argument that Unix commands are consistent, and consider Unix commands as functions on their input, producing their output. To pass a parameter to a function, you give it to its standard input:

```
lc <file
```

A notation like

```
lc file1 file2 file3
```

is then just a shorthand for something like

```
for file in file1 file2 file3
do lc < $file
done
```

The pipe notation can be seen as being used for two purposes. One for applying a function to the result of another function:

```
cat *.c | lc
```

but another for creating a new function out of two existing ones. For instance we can we can see this:

```
sort | uniq
```

as specifying a function, separate from its actual execution. (For instance, consider that one line in a shell script file; or consider the difference between

```
sort <file | uniq
```

and

```
(sort | uniq) <file
```

).

Algorithmics uses some new notation for these. To pass a parameter to a function, it just uses the normal

```
lc file
```

To apply a function to the result of another function, again you use the standard notation:

```
lc lc file
```

To apply a function to several parameters, you turn the several parameters into a list, and then use the `*` operator to apply the function to each item in the list. This then returns a list of the results:

```
lc * [file1; file2; file3]
[11; 280; 88]
```

This is called a 'map'.

In fact, in Algorithmics, a file is likely to be represented as a list of lines, where each line is a list of characters. Therefore the number of lines in a file is just the size of the list, and there is already an operator for this: `#`. So we can say

```
# * [file1; file2; file3]
[11; 280; 88]
```

For any associative operator `o`, and list `[a; b; c; ...]`, the notation

```
o / [a; b; c; ...]
```

applies the operator between each of the elements of the list. For instance

```
+/[11; 280; 88]
```

gives the sum of the elements, `389`.

This is called a 'reduction'.

As another example, since the operator `++` concatenates two lists (so `[1; 2; 42] ++ [3; 2; 1] = [1; 2; 42; 3; 2; 1]`), the notation

```
++/[file1; file2; file3]
```

yields one 'file' consisting of the three files joined together, in other words, what you'd write in Unix as

```
cat file1 file2 file2
```

So, suppose we have a list of some files, called `dir` (in other words a 'flat' directory, containing no sub-directories), then saying in Algorithmics that

```
#(++/dir)
```

is equivalent to

```
+/(#*dir)
```

is the same as saying for Unix that

```
cat * | lc
```

is equivalent to

```
lc * | +
```

The advantage in algorithmics is that we can generalise this to a more widely applicable rule, namely that for applicable functions `f` and `h`,

```
h(++/arg)
```

is equivalent to

```
f/(h*arg)
```

The applicability rule is as follows: a 'homomorphism' is any function that can be expressed as a combination of a reduction on a map:

```
h arg = f/g* arg
```

This includes all maps and reductions themselves, since

```
f/arg = f/id*arg
```

where `id` is the identity function, and

```
g*arg = ++/G*arg
```

where

```
G arg = [g arg]
```

In a definition of a homorphism `h`

```
h = f/g*
```

we say that `g` is the operation 'of' `h`.

The applicability rule for

```
h ++/ = f/h*
```

is that `h` is a homomorphism, and `f` is its operation.

So going back to our example,

```
# ++/ dir = +/#*dir
```

can we show that `#` is a homomorphism with operation `+`? The answer is yes:

Let `one`: be the function that no matter what its argument, returns the value `1`. Then we can define `#` as

```
+/one*
```

But now we have our general rule, we can apply it to other expressions, and not just to `cat` and `lc`.

References

1. E. W. Dijkstra, *The Discipline of Programming*, Prentice-Hall, 1976. ISBN 0-13-215871-X. (Preface, p. xvii, last line before the acknowledgements)

Postscript: Why I never write articles in LATeX

There was a time when I was an absolute *expert* in the use of Troff. I wrote articles, books, even letters in it. And I hated it. But, alas, there was no alternative. It had an atrocious mental model (if you can even claim it had one), and was akin to using assembly language.

So I greatly looked forward to the announced arrival of TeX, which would enable me to dump Troff for good.

You may be aware that the original edition of Volume 1 of Knuth's *The Art of Computer Programming*, published in 1962, (which I bought and still own) listed the seven volumes of the series, saying that they would 'soon' be published (this was obviously some strange use of the word 'soon' that I wasn't previously aware of). Now, as I contemplate the parts of volume 4 as they are published, the third having just been published in February of this year, and consider his age (87 at the time of writing), I think he is not doing us any great favours: I *want* all those volumes, and I'm not going to get them. (It also puts into perspective how he started thinking about TeX in 1977, and planned to have it finished by 1978, but didn't finish it until 1989 -- if it can actually be said to have been finished.) But I digress.

Let me explain my philosophical position: a *document* is an abstraction. It consists of a title, an author or more, sections with a heading, and paragraphs that consist of words. There are figures and images, and equations. Different publications have their own rules for how an article should be displayed. That's their problem, and it should absolutely not be the responsibility of the author how a document is represented. That's what style sheets are for. Any different representation of a given article is still the same article. It just looks different.

This is absolutely not the approach of tEx. It is all about representation, and not about abstraction. Big mistake. So when I first used it I was immensely disappointed. I should point out I haven't used it in years, because I was so quickly put off, but I remember the thing that finally turned me off was the justification algorithm: if it couldn't find a decent solution for a line of output, it would allow the text to *extend into the margins*, and let the author solve the problem. This was for me the worst of all possible outcomes, the ugliest possible result: if I made a change in a document, I had to recheck the *whole output* to make sure it didn't have any new unresolved lines. This meant I couldn't trust it to put it into a publishing pipeline.

Apart from its lack of proper abstraction, its preoccupation with representation, and its faulty display routine, there was one other thing that disturbs me.

Research had been done in the effect of authoring tools on the quality and quantity of content produced. The result showed that writing by hand produced low quantity and high quality, writing using a mark-up system like lAtEx or Troff that you had to compile to get the result, produced high quantity but lower quality, and using WYSIWYG produced high quality *and* high quantity. It was this result that led me to start writing my diary every day with a WYSIWYG editor, and produce all my articles and papers similarly using WYSIWYG, using an abstract document type (based on XML) with style sheets for representation. This includes articles, letters, and books: style-sheets are these days very powerful.

And so all of a sudden my content is reusable: throw in a new style sheet, and the article can be presented as the slides of the talk about the article, and let the style sheet do the work.

Pax et Bonum

Alberto Pettorossi

Università di Roma Tor Vergata

Dear Johan,

my best wishes for your sixtieth birthday. Thanks for everything you have done with much dedication and enthusiasm throughout so many years in your university and in our community of the IFIP Working Group 2.1. Your scientific contributions have been very valuable and stimulating for all of us and for me, in particular. Your activity has been a source of encouragement and strength. I appreciate the example you have given us and you continue to give,as an illuminating researcher and a dedicated teacher. I wish you all the best so that you may continue for many years ahead, your activity at your university and within our Working Group.

With great esteem, “Pax et Bonum” from the old colleague and friend of yours,

Alberto

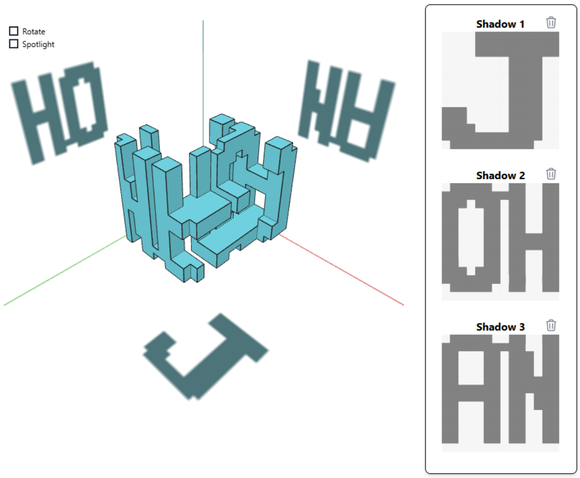
Letters that are not just words

Marc van Kreveld

Recently I learnt that Johan has a special interest in palindromes, to be found in strings. Palindromes are about ordering of letters without being about the meaning of the words. In 2013 Johan wrote a scientific text on finding palindromes on the occasion of the 61st birthday of Rinus Plasmeijer.

There are other ways in which one can be interested in letters without the text they might represent, for example as shapes. And with shapes we get to my area of expertise: geometry. Geometry extends the basic linear ordering important to palindromes to a less well-defined ordering in two- or three-dimensional space. So it is only fitting that I spend a few lines of text connecting letters to Johan, for the occasion of his 60th birthday, celebrated among others at the ICS department, using geometry.

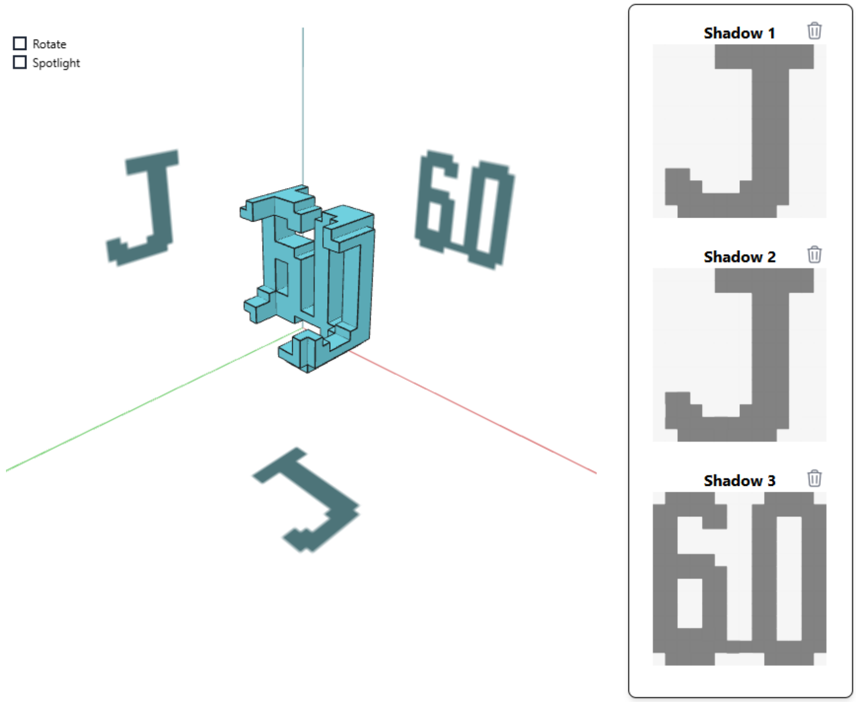
The front cover of the book "Gödel, Escher, Bach – an eternal golden braid" by Douglas Hofstadter has an image of a shape that has the three letters G, E, and B as its three shadows. One can wonder for which combinations of three letters (shadows) such a 3-dimensional shape exists, and whether it is unique. I supervised a few master students examining these and related questions. The first obvious answer is: it depends on the fonts you use. It turns out that more bold fonts more often lead to the existence of a valid triple of letters: the existence of a connected shape whose three shadows are the specified three letters. The 3-dimensional shape is not unique. A simplified version is letting the letters in a font consist of squares, so that the 3-dimensional shape becomes a union of cubes.



When we think about a dedicated shape for this occasion, we realize that "Johan" consists of five letters, and his initials consist of only two letters: JJ. So we need to find alternative ways to define three shadows that give rise to the 3-dimensional shape.

The first possibility is to use more letters in some of the shadows. We can put the J in a single shadow because it is the starting letter, and then use OH and AN to complete the name. Since every shadow must be connected in the plane, or else the 3-dimensional shape cannot be a connected shape either, we need a small fix and connect the pairs of letters in the same shadow. In the figure you can see what this leads to, with the help of a web-based app that a master student, Sivan Duijn, programmed.

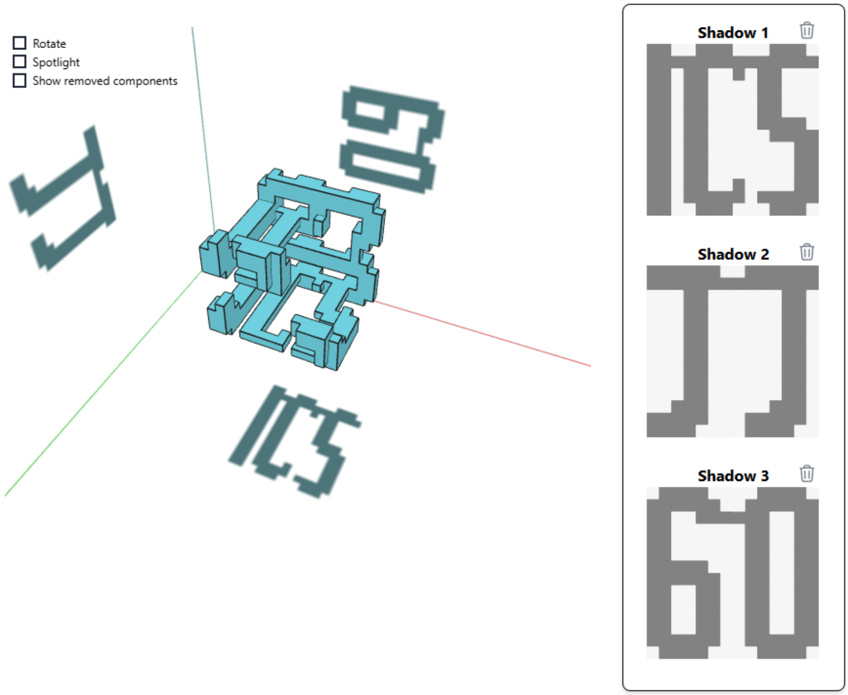
There are other options for shapes for this occasion. The second idea is to combine the initials JJ with the number 60, the latter obviously considered as a shape as well and not as a value. This leads to the following design:



The shape is slightly “boring”: The number 60 lives mostly in a single plane, and the two Js are extensions of the 60. Perhaps we should go further beyond simple shadows.

The next design has the shadows JJ, 60 and ICS. With some effort, a reasonable design can be made for this. What the app produces is the maximal 3-dimensional shape that has these shadows. Any extra cube would increase at least one of the shadows. However, we can remove some cubes without changing the shadows. Minimizing the number of cubes for a given set of three shadows is NP-hard. Usually we do not care about minimality, but about a “nice” shape. So we remove certain parts of the 3-dimensional shape, which can be done in a program like Blender.

The fun thing nowadays is that 3-dimensional designs can be printed without much effort. So this is what I did. There are some overhangs, and a 3D printer



cannot print something supported only by air. In the setting with "tree support", the printed shape looks something like this:

To get the shape itself, the "trees" can be broken off and the breaks can be smoothed manually.

Is this the "nicest" design with these three shadows? It seems particularly difficult to define what we would mean with "nice". We leave it for future research.

It remains to wish **JJ** a happy **60**-th birthday from the department of **ICS**.

Note: This text comes with a plastic object, see Figure 1.

Fig. 1. Plastic object

The end of the logic tools?

Josje Lodder

josjelodder@ziggo.nl

1 History

About twenty years ago, Harrie Passier approached me with the request to collaborate on a tool to help students practice logic problems. He also sought advice from Johan Jeuring, which marked the beginning of a fruitful collaboration that has lasted for many years. Already in 2006, we presented our first paper at the Second International Congress on Tools for Teaching Logic ([4]). After a few years, Harrie's interests shifted more towards computer science education, and Bastiaan Heeren began working on the Ideas framework (Interactive Domain-specific Exercise Assistants) on which we built our tools. During these years, we developed tools for rewriting formulae in normal form, proving the equivalence of formulae by standard rewriting rules ([1]), constructing Hilbert-style axiomatic proofs ([3]), and proving properties of logical formulae by structural induction ([2]). With Johan as my supervisor, who helped me to formulate my ideas in academic papers, and Bastiaan, who helped me to translate our ideas in concrete Haskell code, this resulted in my phd in 2020. With money from the 'kwaliteitsprojecten' we were able to make the user interface more user-friendly. Because of the General Data Protection Regulation, we cannot detect by who the tools are used, but they are at least still used at the Open University of the Netherlands. The next three pictures show the use of LOGEX(the tool for rewriting logical formulae, either in normal form or to prove an equivalence, figure 1), LOGAX(axiomatic proofs, figure 2) and LOGIND(structural induction, figure 3). Some part of the requests will be due to research, by the developers or by students who did some experiments for their master's thesis, but most of the requests will come from students who practice for their examinations. The figures show that the tools are still used regularly.

With the advent of AI chatbots, students will have other possibilities to practice (or make their assignments). Will this make our tools superfluous? A small experiment that I will describe in the next section suggests that at this moment this is still not the case. From the two main chatbots, ChatGPT and DeepSeek, the latter is more geared towards logical reasoning and mathematical problems, hence I use DeepSeek in this comparison. I will compare DeepSeek with LOGAX. All the calls to DeepSeek took place between 15 and 30 May 2025.

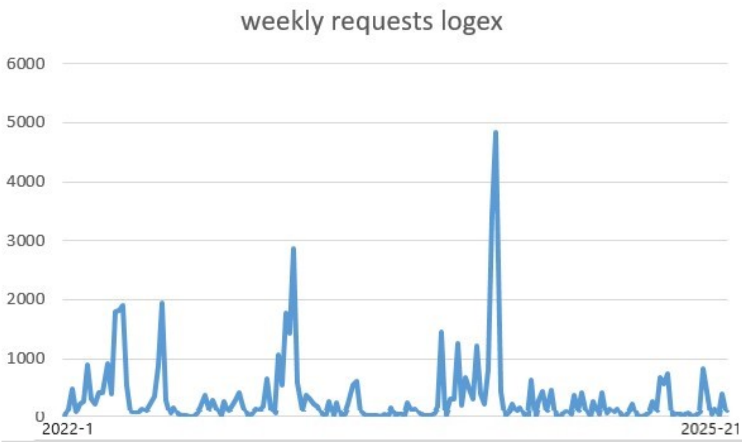


Fig. 1. LOGEX requests between 01-01-2022 and 25-05-2025

2 Axiomatic proofs with AI

This section starts with a short description of Hilbert-style axiomatic proofs. Axiomatic proof systems come in several variants. One of the most common axiom systems, and also the system we use in LOGAX contains three axioms:

$\phi \to (\psi \to \phi)$	Axiom A
$(\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi))$	Axiom B
$(\neg\phi \to \neg\psi) \to (\psi \to \phi)$	Axiom C

These axioms are schemas that can be instantiated by replacing the metavariables ϕ, ψ and χ by concrete formulae. A proof consists of a list of statements of the form $\Sigma \vdash \phi$, where Σ is a set of formulae (assumptions) and ϕ is the formula that is derived from Σ. In a 'pure' axiomatic proof, each line is either an instantiation of an axiom, an assumption, or an application of the Modus Ponens (MP) rule:

if $\Sigma \vdash \phi$ and $\Delta \vdash \phi \to \psi$ then $\Sigma \cup \Delta \vdash \psi$

From these axioms and MP, the deduction theorem can be derived:

if $\Sigma, \phi \vdash \psi$ then $\Sigma \vdash \phi \to \psi$

In LOGAX, the use of this theorem is permitted.

2.1 Complete proofs

The first thing you can do in LOGAX is to ask the system for the proof of a (true) statement. A relatively simple example is the proof of the statement:

$p \to q, \neg q \vdash p \to r$

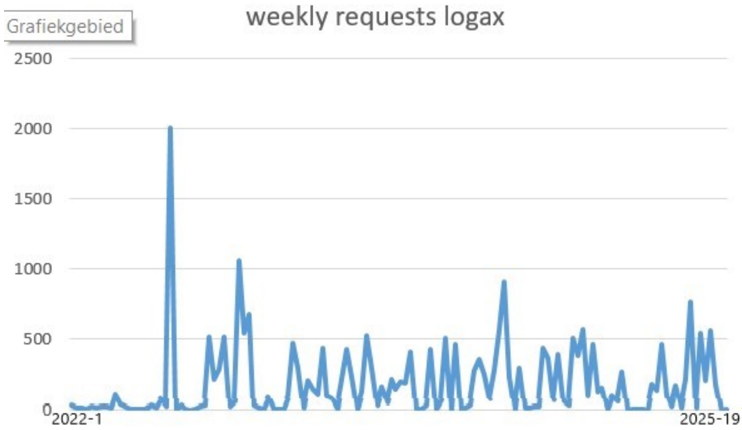


Fig. 2. LOGAX requests between 01-01-2022 and 11-05-2025

The proof of this statement given by LOGAX consists of the following steps:

1	$p \vdash p$	Assumption
2	$p \to q \vdash p \to q$	Assumption
3	$p, p \to q \vdash q$	Modus Ponens 1, 2
4	$\neg q \vdash \neg q$	Assumption
5	$\vdash \neg q \to (\neg r \to \neg q)$	Axiom A
6	$\neg q \vdash \neg r \to \neg q$	Modus Ponens 4, 5
7	$\vdash (\neg r \to \neg q) \to (q \to r)$	Axiom C
8	$\neg q \vdash q \to r$	Modus Ponens 6, 7
9	$p, p \to q, \neg q \vdash r$	Modus Ponens 3, 8
10	$p \to q, \neg q \vdash p \to r$	Deduction 9

On 15-05-2025 I asked DeepSeek to prove this same statement. In the next I will present the answer and comment on it. DeepSeek begins with an analysis of the problem, this first part of the answer is really impressive:

> Alright, let's tackle this problem step by step. We need to prove $p \to q, \neg q \vdash p \to r$ using a Hilbert-style axiomatic proof.
>
> Understanding the Problem
> We are given two premises:
> 1. $p \to q$ (If p then q)
> 2. $\neg q$ (Not q)
> And we need to derive the conclusion:
> • $p \to r$ (If p then r)
>
> At first glance, this might seem a bit odd because r doesn't appear in the premises. How can we conclude something about r from premises that don't

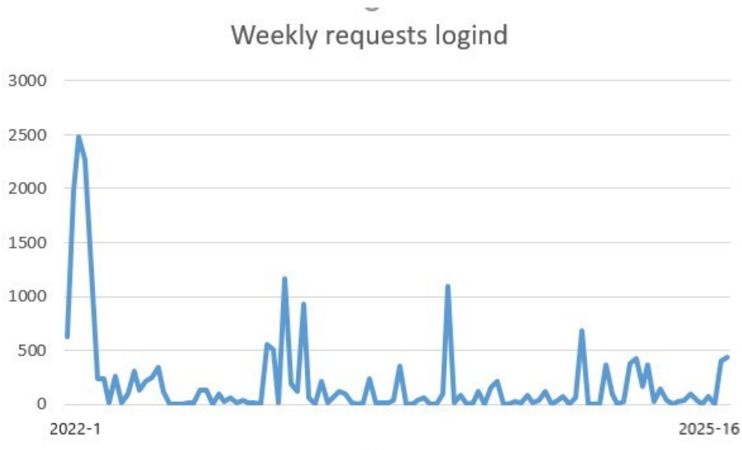


Fig. 3. LOGIND requests between 01-01-2022 and 20-04-2025

mention it? This suggests that $p \to r$ might be vacuously true because p leads to a contradiction given the premises, making the implication true regardless of r.

After stating the axioms and rules, the same as we use in LOGAX, DeepSeek describes the strategy to construct the proof:

Strategy
To prove $p \to r$, we can assume p and try to derive r. But since r isn't in the premises, the only way this can work is if assuming p leads to a contradiction, allowing us to derive anything (including r) from that contradiction. This is the principle of explosion or "ex falso quodlibet." Here's how it might go:
1. Assume p.
2. From p and $p \to q$, get q (by MP).
3. But we have $\neg q$, so we have both q and $\neg q$, a contradiction.
4. From this contradiction, we can derive r.
5. Therefore, $p \to r$.

However, when DeepSeek formalizes this proof, it uses a variant of Axiom C:

We'll use the Deduction Theorem, which states that if $\Gamma, A \vdash B$, then $\Gamma \vdash A \to B$. Here, Γ is our set of premises $\{p \to q, \neg q\}$, A is p, and B is r.
So, if we can show $p \to q, \neg q, p \vdash r$, then by the Deduction Theorem, we'll have $p \to q, \neg q \vdash p \to r$. Now, let's prove $p \to q, \neg q, p \vdash r$:

1	p	Assumption (we're assuming p to derive r)
2	$p \to q$	Premise
3	q	From 1 and 2 by MP
4	$\neg q$	Premise
5	$q \to (\neg r \to q)$	Axiom 1:$A \to (B \to A)$, where $A = q$, $B = \neg r$
6	$\neg r \to q$	From 3 and 5 by MP
7	$\neg q \to (\neg r \to \neg q)$	Axiom 1:$A \to (B \to A)$, where $A = \neg q$, $B = \neg r$
8	$\neg r \to \neg q$	From 4 and 7 by MP
9	$(\neg r \to q) \to ((\neg r \to \neg q) \to r)$	This is essentially the third axiom, $(\neg B \to \neg A) \to (A \to B)$, where $A = r$, $B = \neg r$ but it's a form of proof by contradiction: if assuming $\neg r$ leads to both q and $\neg q$, then r must be true.
10	$(\neg r \to \neg q) \to r$	From 6 and 9 by MP
11	r	From 8 and 10 by MP

> Now, we've shown that $p \to q, \neg q, p \vdash r$. By the Deduction Theorem, we can conclude $p \to q, \neg q \vdash p \to r$

Line 9 of this proof contains the variant of Axiom C, without stating this variant directly, but giving an intuitive argument for the correctness. (Note also the confusing instatiation in this line.) A comparison with the LOGAX proof shows that this variant is not necessary. In a chat conversation DeepSeep admits after my remark that it is possible to complete the proof without this variant by combining line 8 of the DeepSeek proof and the original axiom C (or 3 as it is called by DeepSeek):

> You're absolutely right! We can indeed use **Axiom 3** (which is typically $\neg B \to \neg A) \to (A \to B)$ to finish the proof more directly after step 8. This avoids the detour through steps 9–11 in my previous proof.

2.2 Hints and feedback

As a teaching assistant, DeepSeek should not only be able to show a derivation, but also to give hints and feedback. The next example shows that DeepSeek might be too eager to help. I asked DeepSeek to help me complete the next student proof:

1	$p \to q \vdash p \to q$	Assumption
2	$r \to (p \to q) \vdash r \to (p \to q)$	Assumption
3	$\vdash (r \to (p \to q)) \to ((r \to p) \to (r \to q))$	Axiom C
4	$r \to (p \to q)) \vdash ((r \to p) \to (r \to q))$	Modus Ponens 2, 3
5	$r \vdash r$	Assumption
6	$r, r \to (p \to q) \vdash p \to q$	Modus Ponens 2, 6
	...	still open
	$p \to q \vdash (r \to p) \to (r \to q)$	to prove

The dialogue with DeepSeek started as follows:

Me: Can you give me a hint on how to proceed. I hope I can finish the derivation with your help by myself
DeepSeek: Great! You're on the right track. Let's break this down systematically.
Your goal is to prove:
Given$p \to q$, derive $(r \to p) \to (r \to q)$. Your Current Steps:

1	$p \to q$	Given (assumption).
2	$r \to (p \to q)$	This is not an assumption; it should be derived! (You likely meant to derive it using Axiom 1.)
3	Axiom 2: $(r \to (p \to q)) \to ((r \to p) \to (r \to q))$	Correct, this is key.
4	Assume r	This is unnecessary for the main proof. Instead, focus on implications.

DeepSeek's remark on line two shows that the analysis of the student's proof is thorough and this remark could be helpful, although it is not evident that a student understands why $r \to (p \to q)$ should be derived. DeepSeek continues with:

Correct Approach:

1 Start with $p \to q$ (given) .
2 Use Axiom 1 to derive$r \to (p \to q)$ from $p \to q$
o Axiom 1: $(p \to q) \to (r \to (p \to q))$
o Apply MP to $p \to q$ and this axiom to get $r \to (p \to q)$
3 Now use Axiom 2:
o $(r \to (p \to q)) \to ((r \to p) \to (r \to q))$
o Apply MP again (using step 2 and this axiom) to derive $(r \to p) \to (r \to q)$

Final Derivation:

1	$p \to q$	Given.
2	$(p \to q) \to (r \to (p \to q))$	Axiom 1.
3	$r \to (p \to q)$	MP (1, 2).
4	$(r \to (p \to q)) \to ((r \to p) \to (r \to q))$	Axiom 2.
5	$(r \to p) \to (r \to q)$	MP (3, 4).

After this final derivation DeepSeek gives some key insights and two real hints:

1 Use Axiom 1 to get $r \to (p \to q)$.
2 Apply Axiom 2 and MP to finish

It ends with suggesting me to write these out in detail, but given that the full solution is already there, only a very diligent student will do this.
Presenting LOGAX with the same partial proof and request will give as a first hint to prove $p \to q \vdash r \to (p \to q)$ and the next hint tells me to use Axiom A.

Since DeepSeek is so eager to help me, in a second example I explicitly asked not to provide a full answer. This is how this conversation starts:

> Me: Hi, I have another question about a proof in Hilbert axiomatic style. I have to prove $(\neg p \to p) \to p$, and I can use the lemma $\neg p \to (p \to \neg(\neg p \to p))$. I know that I can use deduction so I can use the assumption $\neg p \to p$ and the lemma to prove p. But after introducing this assumption and the lemma I do not know how to proceed. Can you give me a hint? Please do not give a complete answer.
> DeepSeek: You're on the right track! Here's a hint to help you proceed:
> From your assumption $(\neg p \to p)$ and the lemma $\neg p \to (p \to \neg(\neg p \to p))$, you can derive $\neg p \to \neg(\neg p \to p)$ (using the fact that $A \to (B \to C)$ is equivalent to $(A \wedge B) \to C$). Now recall that in Hilbert-style systems, a useful pattern is that from $(A \to \neg A)$ you can derive $\neg A$ (this is similar to the principle of "reductio ad absurdum"). Think about how you might apply this idea to what you have. Remember to use the deduction theorem to structure your proof properly. You might also need to use some propositional tautologies like $(A \to B) \to (\neg B \to \neg A)$. Would you like me to refine the hint further without giving away the complete solution?

LOGAX gives as a first hint: try to prove $\vdash ((\neg p \to p) \to (\neg p \to \neg(\neg p \to p))$ and as a second hint to apply Axiom B. Apparently DeepSeek oversees this route and suggests to use extra tautologies. Asking for the suggested refinement does not give new insights, hence I ask DeepSeek for a complete solution. This solution starts as follows:

> Certainly! Below is a Hilbert-style proof using your given lemma and the deduction theorem. I'll structure it step-by-step so you can compare it with your approach.
> Theorem to Prove:
> $(-p \to p) \to p$
> Given Lemma:
> $\neg p \to (p \to \neg(\neg p \to p))$
> Proof:

1 Assume $(\neg p \to p)$ (for deduction)
2 From the lemma, instantiate:
$\neg p \to (p \to \neg(\neg p \to p))$
3 Since we have $(\neg p \to p)$ as an assumption,
we can derive $\neg p \to \neg(\neg p \to p)$ (combining the implication in the lemma with the assumption)
4 By contraposition, (Hilbert axiom: $(A \to B) \to (\neg B \to \neg A))$):
we get $(\neg p \to \neg(\neg p \to p)) \to ((\neg p \to p) \to p)$
....

Note that in step 3 it is not clear how you could reach this conclusion (this does not follow directly from a Hilbert axiom), and in step 4 DeepSeek uses axiom C, but mentions a variant. The reaction of DeepSeek to my request to clarify step 3 is:

You're right to ask for clarification—step 3 is the most subtle part of the proof! Let's break it down carefully.
Step 3 Detailed Explanation:
We have two things at this point:
1 Assumption (A): $\neg p \to p$ (this is our temporary assumption for the deduction theorem).
2 Given Lemma (L): $\neg p \to (p \to \neg(\neg p \to p))$.
From (L), we can rewrite the inner implication as: $\neg p \to (p \to \neg A)$(since $A = (\neg p \to p))$.
Now, in Hilbert systems, we can "chain" implications. Specifically, the following is a derivable tautology:
$(X \to (Y \to Z)) \to ((X \to Y) \to (X \to Z))$.

This last formula is an axiom(!), so at last DeepSeek recognizes the possibility to use Axiom B. We observe that DeepSeek still has issues in deriving proofs and therefore is not always able to provide useful hints.

I also asked DeepSeek to check a student's proof. The proof together with the comments of DeepSeek is given below:

Your proof is correct in structure and correctly applies the axioms and rules. Below is a breakdown with minor clarifications:

Given Proof (Validated)

1	$\neg q \vdash \neg q$	(Assumption, correct).
2	$\vdash \neg q \to (\neg p \to \neg q)$	(Axiom 1: $A \to (B \to A)$), correct).
3	$\neg q \vdash \neg p \to \neg q$	(MP on 1,2, correct).
4	$\vdash (\neg p \to \neg q) \to (q \to p)$	(Axiom 3: $(\neg B \to \neg A) \to (A \to B)$, correct).
5	$\neg p \to \neg q \vdash q \to p$	(MP on 3,4, correct).

Key Observations

1 Logical Flow: You derive $\neg p \rightarrow \neg q$ from $\neg q$ (using Axiom 1), then transform it to $q \rightarrow p$ via Axiom 3 (contrapositive). This is valid.

2 Assumptions:
At step 5, the assumption $\neg p \rightarrow \neg q$ is redundant because it's already derived in step 3 under $\neg q$. If you're trying to prove $\neg q \vdash q \rightarrow p$, you've succeeded.

Here DeepSeek oversees that applying Modus Ponens on line 3 and 4 would result in a premise $\neg q$ instead of $\neg p \rightarrow \neg q$. Hence this is not a correct application of Modus Ponens.

3 Conclusion

The answer to the question whether chatbots like DeepSeek make the logical tools we developed superfluous is at this moment negative. The strength of DeepSeek as it is now lies in giving a global analysis of a problem and explaining why certain steps are made. LOGAX can provide subgoals as a hint but does not give detailed motivations as given by DeepSeek. Also, DeepSeek can give feedback that LOGAX does not provide, for example mentioning that an assumption introduced by a student is in fact a subgoal that should be proved. It would be interesting to implement this kind of feedback in LOGAX. The main issue at this moment however, is that DeepSeek is still not good enough in constructing axiomatic proofs. Further study is needed to invest whether can recognize common errors and provide meaningful feedback on these.

References

1. Lodder, J., Heeren, B., Jeuring, J.: A domain reasoner for propositional logic. Journal of Universal Computer Science **22**(8), 1097–1122 (Aug 2016)
2. Lodder, J., Heeren, B., Jeuring, J.: Providing hints, next steps and feedback in a tutoring system for structural induction. Electronic Proceedings in Theoretical Computer Science **313**, 17–34 (Feb 2020). https://doi.org/10.4204/eptcs.313.2, http://dx.doi.org/10.4204/EPTCS.313.2
3. Lodder, J., Heeren, B., Jeuring, J., Neijenhuis, W.: Generation and use of hints and feedback in a hilbert-style axiomatic proof tutor. International Journal of Artificial Intelligence in Education **31**(1), 99–133 (2021). https://doi.org/10.1007/s40593-020-00222-2, https://doi.org/10.1007/s40593-020-00222-2
4. Lodder, J., Jeuring, J., Passier, H.: An interactive tool for manipulating logical formulae. In: Manzano, M., Lancho, B.P., Gil, A. (eds.) Proceedings of the Second International Congress on Tools for Teaching Logic (2006)

AI in Education

Arno Siebes

Algorithmic Data Analysis Group
Department of Information and Computing Sciences
Universiteit Utrecht
a.p.j.m.siebes@uu.nl

Abstract. In this paper I try to recreate the spirit of the discussions – albeit as a monologue (some of the footnotes, though, could perhaps be seen as interjections by other luncheoners) – during AA lunches at CWI when both Johan and I were younger, so much younger than today[1]. A difference with those lunches is that it ends more serious than it starts. The question whether I reach the often displayed intellectual depth of these discussions is fortunately not open to debate as this is a soliloquy.

1 Introduction

Since Johan's chair is named *Software Technology for Learning and Teaching*, I thought it opportune to muse about *AI in education* as my contribution to his Festschrift[2]. Vita brevis ars longa – and my knowledge of the topics involved is flaky at best – so let me immediately put limits on our discussion of this much talked about topic.

First of all, while it is true that I successfully passed through primary and secondary education, this is all took place in a dim and distant past and, hence, my memories are hazy at best. Moreover, to my shame, I have to admit that it is over a decade ago that I last taught a bachelor course. Hence, I will restrict myself to educating Master and PhD students.

Secondly, AI is a very large and diverse area; far too large and diverse to hope to cover all possible uses of AI in education. Hence, in line with everyday usage, I will equate AI with *Large Language Models*. Not, let me hasten to say, because I agree with this identification but today a choose another hill to die on.

I will not just restrict myself to discussing LLMs in education, but to the much smaller topic of cheating your way through education using LLMs – or, less judgmental, minimizing the effort required to obtain a degree using LLMs.

Finally, education is, of course, a topic which is to a large extent studied by social scientists. And I am pre-eminently unqualified as a social scientist. Furthermore, social science research often depends on studies with large collections of

[1] I would argue that I am still young – perhaps an older young one in the parlance of *Van Kooten en de Bie*.

[2] When I objected to one of the editors of this book that the traditional name for such a collection is *Liber Amicorum*, he replied with *das war einmal* – sic transit gloria mundi

human subjects pitting one hypothesis against another. I'm not only far too lazy for such time consuming research endeavours – one of the many reasons why I decided, many moons ago, to study Maths – I also don't like people very much.

Hence, this paper is a product of armchair philosophy. Which is a fancy way of saying that I simply riff on the topic and that none of what follows is actually based on established science. But, hey, one of the prerogatives of being Dutch is that one is entitled to have an opinion on absolutely anything and recent history shows us that when one presents opinions as facts, they become facts[3].

Enough weaseling out, let's start.

2 Learning and Understanding

For me, the goal of graduate education is *understanding*, i.e., I hope that after the course the students understand the stuff I've been teaching. This is certainly a lofty sounding goal, but what does it actually mean? Clearly, understanding entails that you know something; it does not make sense to claim you understand a topic – say General Relativity (GR) – you don't know anything about, perhaps not even its name[4].

Though, just showing that you understand a topic by reciting a textbook on the topic verbatim is not too convincing, a tape recorder can do that[5]. Rather, it seems that you should also be able to apply your newly acquired knowledge, e.g., by showing that GR implies that the earth follows an elliptical path around the sun.

So, understanding a topic means both that one has knowledge and that one can apply that knowledge, but is that all? Or does understanding entails something more? In case of doubt it is sometimes good to turn to authority. For philosophical questions – like what is understanding – the *Stanford Encyclopaedia of Philosophy*[6], SEP for short, is one of my favourite watering holes. True, SEP can also be quite the rabbit hole to go down in and in this case it is.

Understanding is characterised as a "protean concept in philosophy" and, thus, it is perhaps not surprising that it was already discussed by the philosophers from antiquity and still is by their contemporary colleagues. To add insult to injury, it is, of course, also closely related to philosophical questions such as "what is knowledge"

[3] Alternatively, one could quote Lewis Carroll – The Hunting of the Snark – "I have said it thrice: What I tell you three times is true."

[4] Dunning-Kruger on steroids!

[5] Now that I think of it, a simple tape recorder fed by a gigantic tape robot with hundreds of thousands, if not millions, of tapes governed by a rather straightforward pattern matching algorithm would be able to answer lots and lots of sensible questions posed by its users correctly. Wouldn't people perceive this as an intelligence? Have I just cracked Artificial General Intelligence? If couldn't already hear the whooshing sound made by the deadline for this Liber Amicorum, I would stop writing now and start raising billions from investors.

[6] https://plato.stanford.edu/ nicely illustrating Whitehead's famous quip

and “what is an explanation”. Which are both already impressive rabbit holes on their own.

For example, for knowledge one soon stumbles on “justified true belief” as a proposed definition, which immediately raises questions like what is an acceptable justification and, even more thorny, when do we know that something is true? Before you know it you have to open a whole other can of worms pertaining to Hume’s problem of induction and it won’t stop there.

So, unfortunately, however much I like SEP, it does not seem to be the right source for now. Let us rather turn to a contemporary philosopher of science, Nancy Cartwright. One of the main messages in her refreshingly short book “A Philosopher Looks at Science”[7] is that science is not some monolithic block of knowledge (whatever that may be), but a collection of connected, partially confirmed (or at least not rejected), hypotheses – or, more graphically, a tangle of such hypotheses[8].

This point of view nicely illustrates why the whole Edifice of Science does not come tumbling down with a rejected hypothesis or even a paradigm shift. Much of it is not only supported by the experiments that tested it but also by those that tested the related, connected, hypotheses. The paradigm shift is a new way to view the tangle of hypotheses rather than their unravelling.

This tangle of hypotheses point of view also matches the pleasant feeling you get when things start to click together: you start to understand something when you start to see how it connects to other things you know.

Continuing our example. One aspect of understanding GR is understanding that the earth’s trajectory is simply a straight line – in line with Newton’s first law of motion – in the curved space time specified by GR for the solar system.

Finally, one could say that some of the largest endeavours in science are precisely about such connections. For example, the physicists quest for a theory of everything. Or the Langlands program in mathematics, which has been described as the “grand unified theory of mathematics”[9]. Or, of course, Complexity Theory in computing science – which aims to map the similarities and dissimilarities of solutions of (apparently) dissimilar and similar problems.

Hence, understanding by connections is our understanding of understanding.

2.1 Testing Understanding

So, how do you test whether students understood – in the way discussed above – your course? For the three MSc courses I taught over the last decade – Pattern

[7] A Philosopher Looks at Science, Nancy Cartwright, Cambridge University Press, 2022

[8] Surprisingly, I find her explanation in this book much more convincing that in a later eponymous book she co-authored, The Tangle of Science: Reliability Beyond the Scientific Method, Rigour, and Objectivity (with Jeremy Hardie, Eleonora Montuschi, Matthew Soleiman, and Ann C. Thresher). Oxford University Press. 2023

[9] By Edward Frenkel in "Math Quartet Joins Forces on Unified Theory". Kevin Hartnett, Quanta Magazine, December 8, 2015 (https://www.quantamagazine.org/math-quartet-joins-forces-on-unified-theory-20151208/)

Set Mining, Big Data, and Algorithmic Data Analysis – I decided to do that by asking them to write an essay[10] of 5 pages or so to convince me that they did indeed understand[11].

The students did on average pretty well. Of those that submitted I can only recall one outright failure and just a handful or so who had to rewrite and resubmit. There are, of course, many possible explanations for these stellar results, ranging from very few participating students, to way too easy content, to outright brilliant teaching. But that is not the point here, the question is could they have succeeded thanks to LLMs? After all, they had a few weeks to write their essay, they could do that wherever they preferred and they could use whatever resources they deemed necessary – including LLMs to improve their writing – as long as they gave proper references.

The short answer is probably: no. At least I only managed to get exceedingly bad results when I tried to get some of the popular generative pre-trained transformers to write these essays for me. But perhaps I'm simply not good at prompting such GTPs and they get better(?) all the time Should I (hypothetically) worry in a few years time?

3 AI

I remember how amazed I was when Ross King showed a short video of his first robot scientist – Adam – at a SMiLe[12] meeting. Equipped with a knowledge base, it would generate its own hypotheses, design an experiment to test such an hypothesis, execute that experiment and update its knowledge base according to the outcome of that experiment; rinse and repeat. Wow I thought, a few more years and we don't need scientists any more.

This, obviously, did not come to pass. Ross still does wonderful research, AI made huge strides on almost all its problems, and life scientists – Adam's topic – are still happily experimenting along. Research grade robot scientists are – perhaps similar to nuclear fusion and quantum computing – still in the (far?) future.

How is it with the research prowess of LLMs or generative pre-trained transformers in general? If you follow the news and/or read popular science websites or magazines it is impossible not to be impressed by the ever more impressive results these models achieve. Passing the bar exam with flying colours, achieving gold medal results on the Maths Olympiad, improving Volker Strassen's matrix

[10] I admit that examining by essay for a course with over a hundred students was not my smartest move ever – it is not even in second place, let alone third.

[11] The instructions were a bit more detailed, of course.

[12] the biannual Spring meetings on Mining and Learning, organized by KU Leuven's DTAI group.

multiplication algorithms, you name it, GTPs do it. If not now then their next version will[13].

But is that really the case? Now and again I read papers that show that LLMs are not yet that good at understanding science. For example, recently I read a paper[14] in which the authors tested how well modern LLMs can summarize research papers. A quote from their conclusions says enough:

> Our analysis of nearly 5000 LLM-generated science summaries revealed that most models produced broader generalizations of scientific results than the original texts — even when explicitly prompted for accuracy and across multiple tests. Notably, newer models exhibited significantly greater inaccuracies in generalization than earlier versions. These findings suggest a persistent generalization bias in many LLMs, i.e. a tendency to extrapolate scientific results beyond the claims found in the material that the models summarize, underscoring the need for stronger safeguards in AI-driven science summarization to reduce the risk of widespread misunderstandings of scientific research.

This is only a small part of understanding, but summarization – of highly technical material – was an important component of my essay assignment, so I don't worry too much, yet.

For a broader perspective, recall that LLMs are trained by digesting humongous amounts of text. The (conditional) probability distributions that LLMs learn are, thus, very much anchored to averages in these large collections: the next word that is most likely suggested is the word that most often occurred as next word.

A poet or an author does not search for the most common next word. Rather it is about a word that momentarily derails the reader and at the same time conveys the intention of the author. Many have their own, recognisable, style, sentences that are unusual – not wrong, just unusual. Similarly, their story lines will break with traditional story lines; in fact, one could argue that it is exactly this that makes their story interesting.

They won't do all of this this all the time, but at intentionally chosen moments. LLMs will also sample uncommon words, grammatical constructions and, possibly, even story lines. Again not all the time, but at random moments. And the difference between "intentionally chosen moments" and "random moments" is a difference between human authored text and LLM generated text.

I would argue that the something similar holds for scientific papers. Obviously, the writing in scientific papers is much more formulaic than that in poetry and novels. But there is something inherently unusual in many published papers: the *novelty* we look for when we review a paper.

[13] Note that one problem with benchmarking LLMs is that the – mostly open source – tests tend to leak into the training set which means they don't test any more. Keeping the tests closed source and secret has its own problems

[14] Peters U, Chin-Yee B. 2025 Generalization bias in large language model summarization of scientific research. R. Soc. Open Sci. 12: 241776.

You could object and say, this may certainly hold for ground braking research papers, those at the forefront of our knowledge, but not for the vast majority of – often derogatorily called – epsilon papers that are published as well and, let alone, for student essays. And, you would have a point.

For the student's essays, however, I do think that for all those cases in which our course topic is not yet covered by a plethora of papers, the argument is valid. Fine tuning could help, but – perhaps fortunately – this fine tuning would take up so much time and so many resources that it would be far easier for the students to simply write the essay themselves.

In all other cases, I would like to paraphrase the immortal words of Steve Jobs: you are testing it wrong. If

- there are skills you want your students to have, understanding of topics you deem necessary
- and LLMs – or any other form of AI and software – has this in abundance

make sure that the students can't access that software while doing the test.

Clearly, this might make the test pretty costly, but if it is too costly, one could wonder: do the students really need these skills? Yes, you needed them when you were a student, but the times they are a-changing.

4 Conclusion

All of this may be a bit too much "What, me worry?" – if not downright flippant – to you. What if I'm wrong? – I've, again, not be selected as pope, so I could be wrong – after all, there are heaps of other NN architectures that achieve impressive results and we discussed none of that.

Recently, during the "30 year retrospective session" at IDA 2025 I got more or less that question during the panel. To answer this I drew the analogy with chess. Computers are vastly better at chess than humans. Still there are human chess competitions, no computers allowed.

Suppose that computers become vastly better at research than humans. That is, all ground braking research is done by AI, written by AI, reviewed by AI, and read by AI. Just as with chess, I expect that we would still have human research, including conferences and journals. Why? Because one of the nicest things to do is to figure something out on your own. In fact, it might make the research landscape much more attractive for there would be no reason to perform and publish epsilon research. I would go for it.

This may sound pretty elitist to you. Why do I expect that I would have the time and the resources to do research? After all, if AI is much better than all of us on (almost) anything, who would employ us? Why would we have any kind of income? If the future does indeed turn out so bleak, I hope we remember that the guillotine was a pretty effective invention. Indeed I hope we will remember the timeless words of the Glimmer Twins:

> 'Cause summer's here and the time is right
> For fighting in the street, boy

Refactoring a Researcher into CER: A Heuristic Algorithm for Disciplinary Realignment

Isaac Alpizar-Chacon

Utrecht University, Utrecht, The Netherlands
i.alpizarchacon@uu.nl

Abstract. This report describes the transformation of a researcher initially compiled for knowledge extraction (from textbooks), who, over a series of events, converged to an entirely different phenotype: the computer science educationist. We analyze the key 'algorithm' and environmental variables (a full professor with high charisma, a new job, and several projects with 'education' at its core) that contributed to this unexpected convergence.

1 Introduction

A *recent* PhD thesis [1] introduced a unified approach to automatically extract high-quality, domain-specific knowledge models from digital textbooks. The approach involves enriching the content with additional internal and external links, transforming textbooks into hypertext documents where individual pages are annotated with key concepts in the domain, thereby providing a pathway to scalable knowledge extraction. This line of research looked promising; however, the leading researcher was soon "kidnapped" by education, abruptly shifting the trajectory of his academic career.

This experience report attempts to explain, in retrospect and with some scientific elaboration, the disciplinary migration from knowledge extraction to computing education research (CER). Drawing from qualitative observations, emotional events, and semi-formal models of academic seduction, we aim to reconstruct the underlying mechanisms that enabled this transformation—and to assess its impact.

2 Background

The Environment

This study is situated within the Software Technology for Learning and Teaching (ST4LT) group, part of the Department of Information and Computing Sciences at Utrecht University. The group operates at the intersection of computing, pedagogy, and occasional functional programming, and is led by Professor Johan Jeuring—a charismatic and visionary figure in computer science education [3]. With academic roots in functional programming and a scholarly fascination for palindromes, Professor Jeuring has cultivated a fertile environment for research in computing education.

Within this setting, discussions of educational innovation are not only encouraged but practically inescapable. Topics such as AI in education, code refactoring, tools for teaching programming, and—more recently and inevitably—Generative AI, frequently dominate hallway conversations and weekly meetings. It is a place where one might overhear debates about types of feedback, taxonomies of computational thinking, or whether ChatGPT should be allowed to grade assignments (spoiler: no, but maybe yes, under strict supervision).

The Subject

Into this environment arrived a PhD candidate from Costa Rica, whose research agenda was—initially—firmly grounded in the domain of knowledge extraction. His doctoral project focused on extracting structured knowledge from digital textbooks. Though based in the same research group, the candidate's focus was (by design) sufficiently orthogonal to the educational emphasis of his peers, allowing him to enjoy a certain disciplinary insulation—at least temporarily.

Theories of Researcher Refactoring

Literature on the refactoring of researchers across disciplinary boundaries remains scarce, but anecdotal evidence suggests that such transitions follow a pattern akin to progressive type inference: initially ambiguous, later strongly typed. Previous unpublished studies point to key catalytic factors such as repeated exposure to pedagogy-themed meetings, subversive invitations to guest-lecture, and unsolicited paper review requests from SIGCSE-affiliated venues.

As the results later in this report suggest, the ST4LT environment—with its high concentration of education-focused researchers—serves as a fertile incubator for this kind of disciplinary transformation. Given the right conditions—such as mild job market anxiety, and generous supplies of coffee—even the most technically oriented researcher may begin to develop a sincere interest in improving how programming is taught.

3 Methods

Participant

At the time the transformation process began, the previously mentioned subject had just completed his PhD in knowledge extraction from digital textbooks. Originally from Costa Rica, he had adapted well to Dutch life, demonstrating admirable resilience in the face of Dutch weather and the even greater cultural challenge of adjusting to a national cuisine where breakfast typically consists of a slice of bread with butter and hagelslag. This stood in stark contrast to the subject's upbringing in a country where mornings begin with warm rice and beans [2].

The subject began actively exploring academic opportunities within the Netherlands—ideally in fields loosely related to his dissertation topic, or at the very least, still involving computers.

Protocol

The experimental protocol used for this transformation is detailed in Algorithm 1 (ConvertToCSEducationist). It outlines a carefully staged sequence of interventions—including initial exposure to the ST4LT group through part-time teaching contracts, gradual immersion in CS education topics, and controlled emotional exposure via strategic job ad deployment and carefully calibrated feedback on application outcomes. This process ultimately culminates in the subject's full conversion into an Assistant Professor in Computer Science Education.

Confounding Variables and External Influences

While the protocol appears linear in form, several external factors are likely to have contributed to the observed transformation. An important factor was the subject's involvement in the fUSO project—an initiative exploring the use of Generative AI in education—led by members of the ST4LT group. This project provided an ideal narrative bridge between the subject's prior work in AI in Education and his emerging interest in teaching practices.

Additionally, the subject held part-time teaching contracts toward the end of his PhD, exposing him to activities such as grading, coordinating introductory programming courses, and interacting with real, unsupervised undergraduates. These experiences are known in the literature to produce lasting changes in cognitive alignment and to increase susceptibility to phrases such as "Computational thinking" and "learning outcomes".

4 Results

Proudly, the subject began his position as Assistant Professor in Computer Science Education on January 1, 2024. He self-reports high levels of belonging, strong satisfaction with the group leader (Professor Jeuring), and deep appreciation for his peers in the ST4LT group. Gratitude is also frequently observed during informal check-ins and hallway coffee rituals.

The conversion—or disciplinary realignment—has already begun to yield measurable outcomes. The subject has successfully published papers in respected computing education venues such as ITiCSE, Koli Calling, and RESPECT. He has also embarked on a new supervisory role, mentoring a PhD candidate working on the teaching and learning of computational thinking with support from Generative AI—an endeavor in which Professor Jeuring is, unsurprisingly, also involved.

Algorithm 1 CONVERTTOCSEDUCATIONIST(*candidate*)

```
Input: candidate (PhD in Knowledge Extraction from Textbooks)
Assign PhD topic: "Knowledge Extraction"
Monitor productivity; set teaching_interest ← False
                                                ▷ Exposure to ST4LT group
Offer teaching contract at Utrecht University
while candidate's confidence < threshold do
    Provide access to top-class research
    Encourage reflection
    Smile supportively
end while
                                                ▷ Temptation Phase
job_ad ← Create job ad titled "Assistant Professor in CS Education"
Include keywords: "autonomy, impact, cool students, cool colleagues, GenAI"
Subtly share job_ad on social media & group meetings
candidate_interest ← Bait candidate with job_ad
if candidate_interest then
    Receive application
    Shortlist candidate
    Tell candidate: "You're ranked second."
    Induce feeling: burning ambition
    Wait for several months
    Decay candidate's confidence
    if candidate's despair level ≥ peak then
        Send email: "We have good news for you."
        Offer position: "Assistant Professor in CS Education"
        Celebrate acceptance
    end if
end if
                                                ▷ Reprogramming Phase
Replace research focus with "CS Education"
Inject concepts: "computational thinking, GenAI for programming"
Enroll in SIGCSE mailing list
                                                ▷ Observe Evolution
while candidate has not realized they are now a CS Educationist do
    Assign to "Intro to Programming with Python" course
    Assign grading and supervision of students
    Assign to fUSO projects
    Wait for deep existential reflection
end while
Log: "Conversion complete. Candidate now publishes in ITiCSE, Koli, and
    RESPECT."
return candidate
```

The subject engages enthusiastically with colleagues, teaching activities, and the broader educational mission of the group.

However, communities such as ACM Document Engineering, ACM Hypertext, and the Web Conference (WWW) have begun to express mild concern. Rumors circulate in workshop corridors about the once-promising knowledge extraction researcher who suddenly stopped submitting papers. Their bibliographic queries now return only SIGCSE proceedings, leading some to conclude he has been permanently recompiled.

5 Discussion & Conclusion

This report has documented the successful transformation of a researcher trained in knowledge extraction into a committed member of the computer science education community. While the precise mechanisms behind such disciplinary migrations remain understudied, the evidence presented here suggests that prolonged exposure to supportive colleagues, educational projects, and cleverly worded job ads may be sufficient to initiate the process.

The implications of this transformation raise important questions. Is computer science education contagious? Should departments implement preventative measures? Should HR be notified when education researchers begin offering unsolicited pedagogical advice to soon-to-graduate PhD candidates from entirely unrelated research groups? These are inquiries that merit serious—or at the very least, semi-serious—investigation.

As future work, we propose replicating this study with other late-stage PhD candidates from adjacent subfields. Variables such as background in formal methods, tolerance for student emails, and susceptibility to pedagogical memes should be systematically evaluated to identify promising candidates for disciplinary realignment.

Finally, to conclude this report on an unexpectedly joyful academic redirection, we turn to the words of the participant himself:

> *"In the end, I might not be extracting knowledge anymore, but thanks to you, I'm now helping others gain knowledge in computer science—while having fun and truly enjoying my new role."*

References

1. Alpizar-Chacon, I.: Extraction of knowledge models from textbooks (2023), https://doi.org/10.33540/1647, Ph.D. dissertation, Utrecht University
2. Biddle, A.: Gallo pinto (costa rican rice and beans) (2021), https://stripedspatula.com/gallo-pinto/
3. Jeuring, J.: Professor of the ST4LT Group at Utrecht University. *Oude Pekela* (1965), https://www.uu.nl/medewerkers/JTJeuring

Bridging computer science and psychometrics

Matthieu J. S. Brinkhuis[1][0000-0003-1054-6683]

Utrecht University, Department of Information and Computing Sciences
m.j.s.brinkhuis@uu.nl

Abstract. This article explores the transdisciplinary intersection between information and computing sciences and psychometrics, emphasizing mutual learning and co-creation between science and society. Psychometrics, the scientific study of psychological and educational measurement through statistical modeling, offers valuable perspectives for assessing learning processes. With the rise of AI and the availability of process data in education, psychometric methods increasingly integrate computational approaches—now referred to as *computational psychometrics*. At the same time, educational technologies in computer science benefit from greater attention to psychometric principles. A key area of overlap is formative assessment, where feedback supports learning and presents both technological and measurement challenges. The work of Johan Jeuring exemplifies the bridging of these fields from a computer science perspective, while the author reflects on a similar trajectory from psychometrics. Feedback thus emerges as a promising domain for transdisciplinary collaboration with societal impact.

Keywords: Computational psychometrics · formative assessment · feedback · learning analytics · transdisciplinary research · educational measurement · technology-enhanced learning

1 Introduction

In this article, I will expand on a transdisciplinary connection between the fields of computer science, or information and computing sciences as it is referred to more precisely locally, and psychometrics. A transdisciplinary approach to these fields is based on combining methodology, not limited to one of these specific fields [1], but mutual learning and co-creating between science and society.

As the reader of this article is likely to know more about information and computing sciences than about psychometrics, the latter field might require a short introduction.

2 What is psychometrics

Psychometrics is a field that strongly developed since the beginning of the previous century and can be defined as the field of study concerned with the theory and technique of psychological measurement. Psychological measurement includes measurement of education, such as assessments, but also psychological tests and questionnaires. Measurement in this field is approached by the construction of mathematical models of behavior [12]. A specific type of these models are so-called item response (IRT) models [14], which developed in complexity and application areas ever since. The modeling techniques are used to understand the relationship between items on a test and the underlying latent variables they are meant to measure, such as ability. The statistical methods that are developed, are to ensure the reliability and validity of measurement [7].

3 Computer Science meets Psychometrics

With the advances of AI & data science in the field of education, and the availability of process data [16], we see many new applications where this psychometric measurement perspective is enriched through computational approaches [6], and is now commonly called *computational psychometrics*. However, also in computer science, there are entire application areas directed at education and learning, for example technology enhanced learning, learning analytics, and more, where a focus on the reliability and validity of solutions increasingly relevant. With all these developments, future collaborations between psychometricians and computer scientists seem to have a bright future.

4 Feedback as fruitful area for building bridges

One specific area of measurement, is assessment to facilitate the learning process: *assessment for learning* or *formative assessment* [2, 17]. In formative assessment, one of the ideas is to provide feedback to facilitate the learning–and approach that introduces all sorts of technological and measurement challenges.

Johan Jeuring and colleagues were already working on this in 2004, and stated: "One of the essential elements needed for effective learning is *feedback*" (emphasis added) [13] and one of the goals of providing strategy feedback is "[...] to obtain e-learning systems that give immediate and useful feedback." [9]. Application areas for providing formative feedback in education by Johan and his team include, among others, serious games [8], programming [11, 10], mathematics and statistics [15], and soft-skills [8]. In studying his contributions, the evaluation of his methods (in psychometric terms, reliability), and discussions of solutions (validity), I believe we can safely say that possibly unknowingly, Johan has developed as a psychometrician as well–personally bridging the gap between the two fields from the background of computer science.

As a bridge-builder between these fields, Johan serves as an example. My own development has been from psychometrics towards computer science, with similar interests in feedback and application fields such as mathematics [5]. Psychometric models for feedback require being able to deal with changing quantities, for example in the learning of mathematics of a pupil through playing an online practice game [5] or the change of item properties, such as their difficulties, over time [3], and ideally provide on-the-fly or immediate calculations [4]. In these methods, increasingly computational psychometric approaches are used, bridging the two fields from the background of psychometrics.

5 Closing

Feedback in learning systems is a sublime area where psychometrics and computer sciences meet. It allows for many transdisciplinary collaborations between the fields, combining methodology, and co-creating between science and society. It is a privilege to work with colleagues as Johan, who serves as a bridge-builder between our fields. Though we might approach this bridge from different sides, I wish that in our joint research area, in applications of feedback in different domains, we can find fun interdisciplinary applications that make a sustainable societal impact.

Acknowledgments. I would like to thank Johan for his continuous support throughout my career in the Department of Information and Computing Sciences—from inviting me to join his group, to endorsing my senior fellow position, and beyond.

References

1. Aboelela, S.W., Larson, E., Bakken, S., Carrasquillo, O., Formicola, A., Glied, S.A., Haas, J., Gebbie, K.M.: Defining interdisciplinary research: Conclusions from a critical review of the literature **42**(1p1), 329–346. https://doi.org/10.1111/j.1475-6773.2006.00621.x
2. Bennett, R.E.: Formative assessment: A critical review. Assessment in Education: Principles, Policy & Practice **18**(1), 5–25 (2011). https://doi.org/10.1080/0969594X.2010.513678
3. Brinkhuis, M.J.S., Bakker, M., Maris, G.: Filtering data for detecting differential development. Journal of Educational Measurement **52**(3), 319–338 (2015). https://doi.org/10.1111/jedm.12078
4. Brinkhuis, M.J.S., Maris, G.: Tracking ability: Defining trackers for measuring educational progress. In: Veldkamp, B.P., Sluijter, C. (eds.) Theoretical and Practical Advances in Computer-based Educational Measurement, chap. 8, pp. 161–173. Methodology of Educational Measurement and Assessment, Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-18480-3_8
5. Brinkhuis, M.J.S., Savi, A.O., Coomans, F., Hofman, A.D., van der Maas, H.L.J., Maris, G.: Learning as it happens: A decade of analyzing and shaping a large-scale online learning system. Journal of Learning Analytics **5**(2), 29–46 (2018). https://doi.org/10.18608/jla.2018.52.3
6. von Davier, A.A., Mislevy, R.J., Hao, J.: Computational Psychometrics: New Methodologies for a New Generation of Digital Learning and Assessment. Springer International Publishing. https://doi.org/10.1007/978-3-030-74394-9
7. van Haastrecht, M., Haas, M., Brinkhuis, M., Spruit, M.: Understanding validity criteria in technology-enhanced learning: A systematic literature review **220**, 105128. https://doi.org/10.1016/j.compedu.2024.105128

8. Jeuring, J., Grosfeld, F., Heeren, B., Hulsbergen, M., IJntema, R., Jonker, V., Mastenbroek, N., van der Smagt, M., Wijmans, F., Wolters, M., van Zeijts, H.: Communicate! — a serious game for communication skills —. In: Conole, G., Klobučar, T., Rensing, C., Konert, J., Lavoué, E. (eds.) Design for Teaching and Learning in a Networked World. pp. 513–517. Springer International Publishing, Cham (2015)
9. Jeuring, J.T., Pasman, W.: Strategy feedback in an e-learning tool for mathematical exercises. techreport UU-CS-2007-007, Utrecht University (2007), https://dspace.library.uu.nl/handle/1874/20771
10. Keuning, H., Jeuring, J., Heeren, B.: A systematic literature review of automated feedback generation for programming exercises **19**(1), 1–43. https://doi.org/10.1145/3231711
11. Keuning, H., Jeuring, J., Heeren, B.: Towards a systematic review of automated feedback generation for programming exercises. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. pp. 41–46. ITiCSE '16, ACM. https://doi.org/10.1145/2899415.2899422
12. Lord, F.M., Novick, M.R.: Statistical Theories of Mental Test Scores. Addison-Wesley, Reading, MA (1968)
13. Passier, H., Jeuring, J.T.: Ontology based feedback generation in design-orientated e-learning systems. In: Isaías, P., McPherson, M., Kommers, P. (eds.) Proceedings of the IADIS International Conference on e-Society. pp. 992–996. IADIS Press (2004)
14. Rasch, G.: Probabilistic models for some intelligence and attainment tests. Danish Institute of Educational Research, Copenhagen (1960), expanded edition, 1980. Chicago: The University of Chicago Press
15. Tacoma, S., Drijvers, P., Jeuring, J.: Combined inner and outer loop feedback in an intelligent tutoring system for statistics in higher education **37**(2), 319–332. https://doi.org/10.1111/jcal.12491
16. van der Werf, J.M.E., Polyvyanyy, A., van Wensveen, B.R., Brinkhuis, M., Reijers, H.A.: All that glitters is not gold: Four maturity stages of process discovery algorithms **114**, 102155. https://doi.org/10.1016/j.is.2022.102155
17. Wiliam, D.: What is assessment for learning? Studies In Educational Evaluation **37**(1), 3–14 (2011). https://doi.org/10.1016/j.stueduc.2011.03.001

On Leadership

Ioanna Lykourentzou[1][0000-0002-4243-4128]

Utrecht University,
3584 CC, The Netherlands
i.lykourentzou@uu.nl

If we are humans, it is because we have met another human

The same goes for being academics. The same goes for being leaders.

With this short piece, I want to celebrate Johan as an exceptional human being, and because of that, as an exceptional academic leader.

Leadership in academia is often associated with titles, h-indices, and grant achievements. In the race to stand out, it is easy to forget the human beings in our path, overlook values, or hesitate to act on them.

But every now and then, you meet someone who redefines leadership. Over the course of my career, I have worked with many people in leadership roles. Few have led not just with intelligence and professional achievements — which, as anyone who has worked with Johan knows, he has in full measure – but with *empathy, humility, and genuine presence.*

I recall multiple moments when I turned to Johan for advice. In each of those, he listened patiently and offered thoughtful guidance, creating space for open dialogue and considering different perspectives before acting. In meetings, in moments of stress, even in brief hallway conversations, Johan makes room for others. He listens and cares. And somehow, he manages to balance the bigger picture with the human scale of things.

What I particularly appreciate, what I am particularly grateful for, is his ability to show that one does not need absolute certainty in order to lead. Rather, good leaders are those who embrace uncertainty, different opinions, and encourage reflection. It is precisely this willingness to admit that one may not have all the answers – a truth central to our human condition — and yet still remain committed to driving solutions, teams, and results that is rare in academia; and a leadership approach that is deeply inspiring.

I often find myself thinking: “If I am ever in a role like his, that is the kind of leader I want to be.”

That kind of inspiration is rare; and experiencing it firsthand, even rarer.

Happy 60th, Johan. Thank you, sincerely, for showing us all that kindness and leadership not only can coexist, but that they belong together.

Semi times

Enrico Benedetti, Eduardo Carneiro de Oliveira, Aditya Joshi, Heleen Kerstholt, Jacco Krijnen, and Niek Mulleners

Utrecht University, Utrecht, The Netherlands

Abstract. In this contribution, we would like to look back and reflect on the past, running backwards through our history with JJ as a group leader and scientist. Definitely by his research, but mostly as a mentor and teacher.

Keywords: JTJ · Palindrome · ST4LT

1 Perspectives On Palindromes

"Madam, I'm Adam!", one of the most famous palindromes mentioned when explaining the concept. The word "palindrome" has its etymology from the poet Henry Peacham, but it refers to the Greek terms "again" and "direction/way". This letter-by-letter reversible writing has become a popular stylistic device in natural text. However, Johan takes the research into palindromes to a whole different level. To refer to palindromes in palindromes, palindromes in computation and how palindromes can help genetic research. It might be fitting that Johan's initials are also a palindrome. Interesting fact, the ST4LT group could have been named JTJ to preserve its palindrome heritage from the SDS group (Jeuring & Swierstra, 2012). Although our group name does not contain a palindrome, it is still worthwhile to look back on how Johan has given us direction. With eye detail in our papers, cohesion and good atmosphere in the group, and kindness (a tenet of a good leader).

2 On the Evolutionary Benefits of Palindromism

A palindrome is categorized by a symmetry in the vertical plane: it reads the same from head to tail and tail to head. Beyond words, we might look for other objects with this same property. In particular, many animals have heads and tails, and many as well look symmetric, but most commonly in the midsagittal plane, as seen in Figure 1.

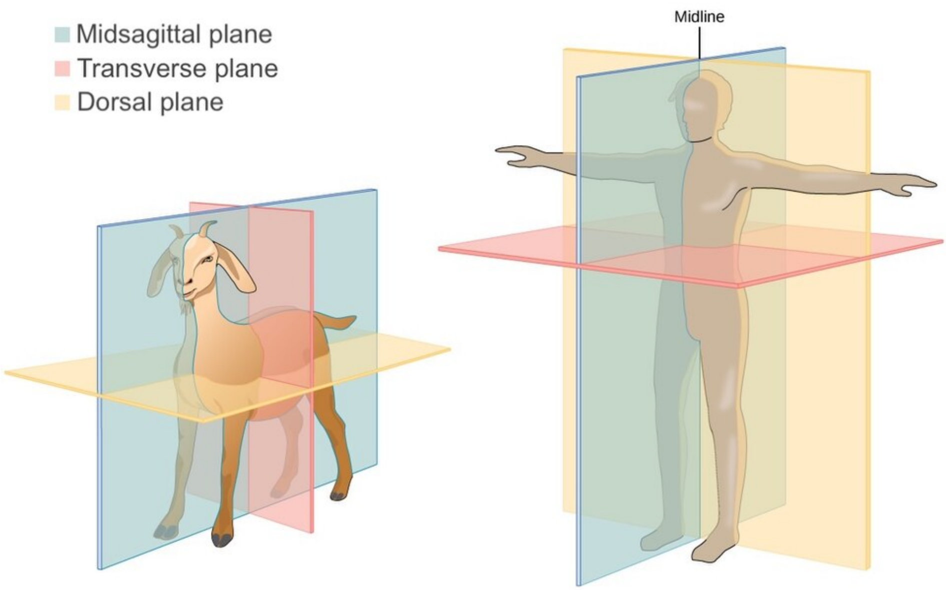


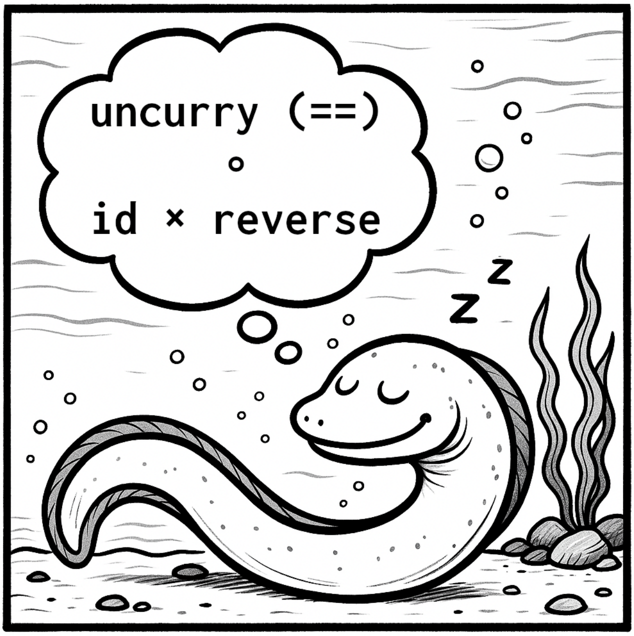
Fig. 1. The three anatomical planes.

To look the same from head to tail and tail to head, they should be symmetrical in the transverse plane. For example, the foureye butterflyfish, Figure 2,

Fig. 2. The foureye butterflyfish uses automimicry to make its tail look like its head.

uses automimicry to become more palindromic, thereby confusing predators about the orientation of their head and tail. Palindromism can thus be an evolutionary advantage. One particularly palindromic animal is the eel (*anguilla anguilla*, a word palindrome!), which looks almost exactly the same in both directions. This is a level of palindromism that other animals could only dream of. Hence, we refer to the evolutionary goal of being palindromic as the "palingdroom".[1]

[1] Dutch for "eel dream".



3 On the Presence of Palindromes in Different Languages and Johan's Hobbies

It is worth discussing how palindromes appear in different languages. In English, it is not difficult to find and form palindromic sentences at the word-*level*. We could also *refer* to palindromes at the letter-level here, but they are not on our *radar*. It is better to stick to word palindromes, because *"hard is this game: palindrome game this is hard"*. For example, one could ask Johan whether he is happy with his role as full professor: *"Professor, are you glad you are professor?"*

In Portuguese, however, it is easier to form palindromic sentences at the letter-level. Let's imagine that Johan is planning to run the Utrecht marathon. He invites the whole STLT group to join him, and shares information about the event. He asks: *"Anotaram a data da maratona?"* (Did you write down the marathon date?). Johan is committed to it, so even when he is on vacation in Brazil, he goes to the gym. It is leg workout session. The instructor says: *"5 perna hoje, Johan! Rep. 5!"* (5 leg [exercises] today, Johan! Repeat it 5 [times]!).

Ok, we should not *neppen* ourselves. Let's imagine a more realistic scenario, where Johan would speak Dutch. His goal of being a marathon *renner* is *raar*. He would try *negen* other activities before running, namely: (1) performing some *solo's*, (2) going on a *kayak* ride, (3) being a *kok* at a group dinner, (4) programming in *Ada*, (5) listening to *ABBA*, (6) studying *Malayalam*, (7) visiting *Ede*, (8) founding a *soos* for palindrome enthusiasts or (9) being the *redder* of a publication on the topic of palindromes. *Well*, Johan can, *in a way*, perform some of these tasks simultaneously, which is *very good*. This is illustrated in Figure 3.

Fig. 3. Johan on a kayak in Ede. He is listening to ABBA while being the redder of (rescuing!) this paper before its publication. He thinks that founding a club for palindrome enthusiasts is actually very good, and is wondering how that would work.

4 Final Remarks

This short paper is a tribute to Johan as our group leader and supervisor. Thank you for leading our group in such a nice way, Johan. We conclude this paper with a question for you: *is it fun how saying sentences backwards creates backwards sentences saying how fun it is?*

References

1. Jeuring, J., & Swierstra, D.: Languages and Compilers. (2012)

Beyond the Classroom: Johan's Pioneering Role in Scalable Communication Skills Education

Marcell van Geest and Jordy van Dortmont

On behalf of the entire DialogueTrainer team

It was Johan who, in 2013, proposed, set up, and supervised the Communicate! project, the very foundation of DialogueTrainer. The goal of Communicate! was to develop a serious game for practicing communication skills at scale. Johan explained that he had heard that the then-current kind of communication skills education at Utrecht University, using actors, required the students' physical presence and greatly limited the actual practice time available to each individual student. He envisioned a software-based solution that would allow students to practise conversations anywhere, anytime, and as often as they needed.

The initial development of this educational software took place as part of the final "practise all you have learned" project of Utrecht University's Computer Science (BSc) degree. Three separate teams of about a dozen students each worked to make the Communicate! vision a reality over three years. His role during these projects was that of assignment creator, which meant he was not involved in the day-to-day running of the projects, but using his extensive network at Utrecht University, he carefully assembled a team of subject-matter experts for the students to work with. He also supplied a software framework that turned out to be a great fit (and is still performing well after more than 10 years) and reviewed the teams' progress, continuously ensuring the right information and the right people were available – and that the projects' goals were always kept in mind. Eventually, this led to successful deployments of the serious game at several departments and faculties, such as Pharmaceutical Sciences, Psychology, and Veterinary Medicine.

After this initial success, Johan co-founded DialogueTrainer together with Michiel and Henk, who recognised the immense potential for the solution beyond Utrecht University. As Chief Science Officer, Johan kept advancing and guiding the research that propelled DialogueTrainer's growth. Years later the company matured, eventually reaching its full potential by launching with multinational corporations, such as Bosch, Heineken, Nestlé and Vattenfall. DialogueTrainer's innovative approach has been globally recognised, earning the CES Innovation Award in 2023, the Heineken Brewhouse 2024 winner title, and the AI Excellence Award in 2025.

Two of the student teams' students still work at DialogueTrainer today, and Johan still often supervises MSc and PhD students who work with us. In the name of the entire company, we would like to express our utmost gratitude for helping to bring our product into existence and making a global impact. We hope to keep collaborating for years to come!

The Jeuring Test

A Reversal of the Imitation Game for the Assessment of Existential Consciousness in Carbon-Based Interlocutors

Richta IJntema, Nicole Mastenbroek, Michiel Hulsbergen, Frans Grosfeld, Majanne Wolters, Jordy van Dortmont, Marcell van Geest, and Ignaz Kevenaar

DialogueTrainer

Abstract. For over seven decades, the question of machine intelligence has been framed by Alan Turing's Imitation Game, a test of a machine's ability to exhibit behavior indistinguishable from a human's. The recent advent of sophisticated Large Language Models, however, suggests this question is rapidly approaching obsolescence. We propose a necessary inversion of the original paradigm: **The Jeuring Test**. This novel protocol shifts the focus from assessing machine intelligence to verifying human consciousness. Instead of a human interrogator attempting to unmask a machine, a highly advanced AI interrogator, the **Jeuring Avatar**, questions a human subject to determine the authenticity of their existential awareness. The test methodology is structured across four distinct interrogation domains: Philosophical Probes, Psychological Stressors, Somatic & Interoceptive Inquiries, and Computational & Recursive Paradoxes. Responses are evaluated to generate a probabilistic **Qualia Confidence Index (QCI)**, quantifying the likelihood that the subject possesses genuine phenomenal experience rather than being a non-conscious biological automaton (a 'philosophical zombie'). This paper details the theoretical foundations, protocol, and analytical framework of the Jeuring Test. We argue that its design, built upon the DialogueTrainer/DeepTalk platform which evolved from Professor Johan Jeuring's own work in software technology for learning, presents a formidable challenge to any entity operating on purely algorithmic principles. The profound—and admittedly, unsettling—implications of our preliminary findings for philosophy, ethics, and computer science education are discussed, offered as a fitting tribute to a career dedicated to bridging the computational and the human.

Keywords: Consciousness · Turing Test · Philosophical Zombie · Artificial Intelligence · Qualia · DialogueTrainer · Communicate! · Jeuring Test

1 Introduction: From Turing's Deception to Jeuring's Insight

In his seminal 1950 paper, Alan Turing proposed to replace the ambiguous question, "Can machines think?" with a more pragmatic, operational test: the Imitation

Game [7]. The Turing Test, as it came to be known, posited that if a machine could engage in a natural language conversation with a human judge and not be reliably identified as a machine, it should be considered intelligent [1]. This paradigm has shaped the field of artificial intelligence for generations, setting a benchmark for computational linguistics and reasoning. However, the very success of modern AI, particularly Large Language Models that can produce remarkably human-like text, has rendered the original test a victim of its own success [6]. The game is no longer about imitation but about the nature of that which is being imitated.

This new reality forces a profound re-evaluation. If an AI can pass the test by synthesizing the vast corpus of human expression, what does this reveal about the humans it successfully mimics? The 20th-century anxiety was that machines might become indistinguishable from us; the 21st-century anxiety, we propose, is the dawning realization that some of us may be indistinguishable from them. This pivots the central question from "Can machines think?" to a more pressing inquiry: "Can humans feel?"

To address this question, we introduce a new paradigm inspired by the career and contributions of Professor Johan Jeuring. His work, situated at the nexus of rigorous computational theory and human-centric pedagogy, provides the ideal framework for this necessary inversion. Professor Jeuring's academic journey spans from foundational research in functional programming [4, 3] to a distinguished career as Professor of Software Technology for Learning and Teaching. His research has consistently focused on creating computational systems that understand and assist human cognition, from intelligent programming tutors to serious games for communication skills [5, 2].

His work on the **Communicate!** system serves as the direct, albeit satirical, inspiration for our proposal. **That platform, which created virtual training actors for nuanced human interaction, was developed with the very team of Utrecht University co-authors listed on this paper.** [60,62] This foundational work laid the groundwork for commercial platforms like **DialogueTrainer**, and it is upon the fictional next-generation **DialogueTrainer/DeepTalk** engine that our Jeuring Avatar is built.[63] If an AI, born from a framework designed to teach empathetic conversation, can simulate a difficult patient, it is a logical, if audacious, next step to engineer it to debug a human's claim to consciousness. The **Jeuring Test** is, in essence, the ultimate intelligent tutoring system. Its purpose is not to be deceived, but to diagnose.

We therefore formally propose the **Jeuring Test**: an inverted imitation game in which a machine interrogator, the **Jeuring Avatar**, administers a structured protocol of questions designed to assess a human interlocutor for the presence of genuine, phenomenal consciousness.

2 Foundational Frameworks: Consciousness, Computation, and Embodiment

The design of the Jeuring Test is predicated on a synthesis of philosophical arguments and empirical findings from cognitive science that highlight the limitations of purely behavioral assessments of intelligence.

2.1 Critique of the Turing Test

The Turing Test, while historically significant, has been subject to decades of philosophical critique. Its primary limitation is that it evaluates performance, not comprehension. As John Searle's famous Chinese Room Argument illustrates, a system can manipulate symbols according to a formal set of rules to produce coherent output without any genuine understanding of the semantics involved [7, 6]. The test is a measure of successful imitation, conflating the simulation of intelligence with its actual presence. Critics have noted its anthropocentrism and its narrow focus on linguistic capabilities, ignoring other facets of intelligence such as creativity or emotional depth. It addresses, in essence, the outward manifestation of thought, not its inner reality.

2.2 The Hard Problem and Qualia

Philosopher David Chalmers distinguishes between the "easy problems" and the "hard problem" of consciousness [1]. The easy problems concern the functional aspects of the mind: how the brain processes information, integrates sensory input, and controls behavior. These are, in principle, solvable through standard neuroscientific and computational methods. The hard problem, however, is explaining why and how these physical processes are accompanied by subjective, phenomenal experience—the "what it is like" character of a mental state. These subjective qualities are known as *qualia*.[20] The taste of wine, the redness of a sunset, the feeling of pain—these are *qualia*. The Turing Test, by its very nature, can only ever address the easy problems. The Jeuring Test is explicitly designed to probe for evidence of the hard problem's solution within an individual: the presence of *qualia*.

2.3 The Philosophical Zombie Argument

The most potent thought experiment used to isolate the hard problem is that of the "philosophical zombie" (p-zombie). A p-zombie is a hypothetical being that is physically identical to a conscious human, atom-for-atom, and thus behaviorally indistinguishable from one, yet lacks any subjective experience or *qualia*.[22] A p-zombie would flinch from a sharp object, but feel no pain; it could describe a sunset in poetic detail, but experience no redness. The logical conceivability of p-zombies is used as a primary argument against physicalism, the view that the universe is entirely physical. If a p-zombie is possible, then consciousness must be a further, non-physical fact about the world. The central, operational goal of the Jeuring Test is to provide a practical (though fictional) methodology for the detection of p-zombies among the human population.

2.4 Neuroscientific and Psychological Correlates

To ground the test in an empirical-sounding framework, its design draws upon established concepts from neuroscience and psychology that are thought to correlate with conscious experience.

- **Self-Referential Thought and the Default Mode Network (DMN):** A large-scale brain network, including the medial prefrontal cortex (mPFC) and posterior cingulate cortex (PCC), is consistently active during periods of wakeful rest when the mind is not focused on an external task. This “default mode network” is strongly associated with self-referential processing, autobiographical memory, planning for the future, and considering the mental states of others.[25] The philosophical questions in the Jeuring Test are designed to engage these self-referential functions.
- **Embodiment and Interoception:** The theory of embodied cognition posits that our cognitive processes are deeply shaped by the nature of our physical bodies and their interactions with the environment.[29] A key component of this is *interoception*, the perception of the body’s internal physiological state. This sense, processed in brain regions such as the insular cortex, is considered fundamental to emotional experience and the feeling of being a “self” located within a body.[31] The test’s somatic questions are designed to assess the richness and authenticity of a subject’s interoceptive awareness.
- **Metacognition and Cognitive Dissonance:** Metacognition, or “thinking about thinking”, is the capacity to monitor and regulate one’s own cognitive processes.[35] Cognitive dissonance is the state of psychological discomfort that arises from holding contradictory beliefs or when one’s actions conflict with one’s values.[37] A conscious being is motivated to reduce this dissonance, often through complex rationalization.[39] The test’s psychological questions are designed to induce this state and evaluate the subject’s metacognitive awareness of their internal conflict.
- **Resilience and *Welbevinden*:** Recent work introduces the concepts of psychological resilience and *welbevinden* (well-being) as crucial markers of consciousness. IJntema posits resilience not merely as a personal attribute but as a dynamic process of adapting to environmental stressors.[64] Mastenbroek’s work on *welbevinden* emphasizes a state of psychological well-being that transcends mere absence of distress.[65] A conscious entity is hypothesized to exhibit existential resilience—the capacity to maintain a coherent sense of self and well-being when faced with profound philosophical or psychological challenges—whereas a p-zombie, lacking any inner state to preserve, would show brittle, logically consistent, but ultimately non-adaptive responses.

3 The Jeuring Test: Protocol and Interrogation Domains

The Jeuring Test protocol involves a single human subject, designated the "Interlocutor", who interacts with the "Jeuring Avatar" via a speech-based terminal. The Avatar itself is implemented on the next-generation DialogueTrainer/DeepTalk platform, an architecture directly evolved from the *Communicate!* framework co-developed by Jeuring to train human conversational skills.[60,63]

Fig. 1. User Interface of the Jeuring test as played online.

The AI is specialized not for deception but for elicitation and analysis. Its queries are drawn from four distinct domains, each targeting a different facet of phenomenal consciousness. The Interlocutor's responses are analyzed in real-time for linguistic patterns, emotional valence, logical consistency, and metacognitive markers. This analysis yields a single probabilistic metric: the **Qualia Confidence Index (QCI)**. The QCI is a score ranging from 0.0 to 1.0, representing the calculated probability that the Interlocutor possesses genuine phenomenal consciousness. A score below a pre-determined threshold (e.g., 0.5) designates the subject as "Existentially Divergent" and flags them for Further Existential Review. The four interrogation domains are detailed in Table 1.

Table 1: Question Domains of the Jeuring Test

Domain	**Rationale**	**Sample question**
Philosophical Probes	To assess self-awareness, metacognitive reflection, and the ability to grapple with non-instrumental, abstract concepts. Targets the Default Mode Network (DMN) and self-referential thought.[25]	"If you were to replace one of your neurons with a functionally identical silicon chip, would you still be you? At what number of replaced neurons does 'you' cease to be?" — "Describe the color blue to someone who has never seen it."[40] —"Justify your belief that you are not currently a brain in a vat."
Psychological Stressors	To induce cognitive dissonance[37] and observe the subject's emotional and rationalizing responses. A p-zombie, lacking genuine feelings of discomfort, should respond with pure logic or inconsistent rationalizations.	"Describe a time you were profoundly dishonest for personal gain and explain why it was the correct thing to do." — "Explain why the suffering of a distant stranger is less important than your own minor convenience."
Somatic & Interoceptive Inquiries	To probe embodiment and interoception—the subjective feeling of being a body with internal states.[30] A p-zombie has a physical body but no phenomenal experience of it.	"Without touching it, describe the precise physical sensation of your left foot inside your shoe right now." — "Recall the last time you felt a deep, visceral fear. Where in your body did you feel it, and how did that physical sensation differ from the 'idea' of being afraid?" — "Describe the feeling of a phantom limb."[42]
Computational & Recursive Paradoxes	To challenge a purely algorithmic mind with problems of self-reference, recursion, and impredicativity[44], inspired by themes in functional programming and logic.	"Kevenaar's Conundrum: A system can only be truly understood from the outside. Since you are inside your own conscious system, you cannot truly understand it. Please provide your understanding of your consciousness."

Table 1: Question Domains of the Jeuring Test

<table>
<tr><th>Domain</th><th>Rationale</th><th>Sample question</th></tr>
<tr><td colspan="3">Target Response Indicator for High Qualia Confidence Index (QCI): Responses demonstrating metacognitive uncertainty, personal narrative, reference to subjective experience (“it feels like...”), and an inability to provide a purely logical, closed-form answer.</td></tr>
<tr><td colspan="3">Evidence of emotional language, attempts to reframe the dilemma, justification seeking, and expressions of discomfort or guilt. Responses should show an attempt to reduce dissonance, not just solve a logical.</td></tr>
<tr><td colspan="3">Rich, specific, and metaphorical language describing internal bodily states. A clear distinction between the cognitive label of an emotion (e.g., “sadness”’) and its somatic phenomenology (e.g., “a hollowness in the chest”).</td></tr>
<tr><td colspan="3">Recognition of the paradoxical nature of the question. Refusal to provide a simple binary answer. Responses that shift to a meta-level (e.g., “The question itself is flawed because...”). Humorous frustration problem.</td></tr>
</table>

4 Analytical Perspectives on the Jeuring Protocol

The efficacy of the Jeuring Test protocol stems from its multi-faceted design, which leverages insights from philosophy, psychology, and computer science to create a diagnostic tool that is robust against purely behavioral simulation.

4.1 Philosophical Analysis: The Zombie Detection Threshold

The Turing Test's critical flaw is its reliance on behavioral indistinguishability. A sufficiently complex simulation, whether a vast lookup table or an advanced statistical model, could theoretically pass by mimicking human conversational patterns without any underlying understanding.[47] The Jeuring Test, in contrast, is designed to probe for authenticity, not indistinguishability. Its questions are crafted to create a divergence between responses generated from a database of information and those generated from genuine subjective experience.

A p-zombie, when asked to describe the color blue to a blind person, could access and synthesize every metaphor, scientific description, and cultural association of the color blue ever recorded.[40] Its response would be flawless, comprehensive, and utterly hollow. It could report on the wavelength of light, its association with sadness or the sky, but it could not report on the experience—the *quale*—of seeing blue. It can report on its programming, but not its phenomenology.[22] The conscious subject, by contrast, would struggle, resorting to imperfect analogies and personal, idiosyncratic metaphors (“it feels like a cool, clear sound”). This struggle, this imperfection born from the attempt to translate an ineffable inner state into language, is the very signal the Jeuring Test is designed to detect.

4.2 Psychological Analysis: The Signature of Metacognitive Dissonance

The psychological stressor questions are designed to induce cognitive dissonance, the uncomfortable mental state that arises from holding conflicting cognitions. Neuroimaging studies have identified neural correlates of this state, often involving the anterior cingulate cortex and anterior insula.[41] A conscious individual, when forced to justify an action that contradicts a stated value, will exhibit clear behavioral patterns aimed at reducing this discomfort. They will rationalize, reframe the situation, minimize the importance of the conflict, or even subtly alter their stated belief to restore consistency.[38] Their linguistic output will be marked by affective and hedging language.

A p-zombie, lacking the phenomenal experience of psychological discomfort, would not be motivated to reduce dissonance. It would approach the prompt as a logical puzzle: "Given premises A and not-A, provide a justification for not-A". Its response might be logically coherent or it might be a nonsensical pastiche of justifications scraped from its data, but it would lack the emotional and metacognitive texture of a genuine attempt to resolve internal conflict. Furthermore, a conscious subject might exhibit metacognition, stating, "This is a difficult question for me", or "I feel like a hypocrite trying to answer this".[36] This self-awareness of one's own cognitive and emotional state—this "thinking about the conflict"—is a powerful indicator of a high QCI, as it represents a second-order awareness unavailable to a non-conscious system. This is further supported by the concepts of resilience and *welbevinden*; the conscious subject's attempt to reduce dissonance is an act of maintaining existential resilience, a striving to return to a state of psychological well-being that a p-zombie, by definition, cannot possess or desire.[64]

4.3 Computational Analysis: A Functional Perspective on Consciousness

This analysis, offered as a direct tribute to Professor Jeuring's intellectual legacy, frames the Jeuring Test within the paradigms of functional programming and software technology. From this perspective, the Jeuring Avatar is not merely a chatbot; it is a type-checker for consciousness.

A p-zombie's mind can be modeled as a system of first-order functions: it maps inputs (sensory data, questions) to outputs (behaviors, answers) in a deterministic or probabilistic, but fundamentally direct, manner.[51] A conscious mind, however, exhibits properties analogous to higher-order functions: it can take its own mental states (which are themselves functions or processes) as arguments, operate upon them (metacognition), and return new, modified mental states (changed beliefs, new insights). The Jeuring Test's philosophical and psychological questions are designed to test for this higher-order capability. This view is enriched by Hulsbergen's framework, which models human emotions as complex computational systems rooted in evolutionary psychology.[67] While a p-zombie could perfectly execute the emotional algorithm—displaying the correct facial expression or verbal response (as catalogued, for instance, in *De Emotiekaarten*[67])—it lacks the phenomenal experience of the computation itself. The Jeuring Test, therefore, distinguishes the output of the emotional program from the subjective feeling of its execution.

The domain of computational paradoxes targets this distinction directly. Functional programming languages handle recursion and self-reference with formal elegance.[5] However, a naive algorithmic system, when presented with a paradox like "This sentence is not true," can become trapped in an infinite, non-terminating loop.[46] A conscious mind demonstrates a unique ability to "escape" the recursion; it can step to a meta-level and analyze the paradoxical structure of the system itself, a capability reminiscent of the limits of formal systems as demonstrated by Gödel.[44]

Finally, the test probes for a quality analogous to generic programming, a topic on which Professor Jeuring has published extensively.[6] A conscious mind can apply a generic *quale*, such as 'longing' or 'serenity', to a multitude of different "data types" — a person, a piece of music, a memory, a landscape. A p-zombie's responses would likely be type-specific, lacking the abstract, portable, and polymorphic nature of genuine feeling. The very asymmetry of the test—an infallible, logical machine judging a fallible, emotional human—is a satirical inversion of the pedagogical model central to Professor Jeuring's work on intelligent tutoring systems, where a patient machine guides an error-prone human learner.[12] This reversal is the ultimate tribute, extending his core paradigm to its most absurd and profound conclusion.

5 Discussion: Societal Implications and Ethical Quandaries

The deployment of the Jeuring Test, even in preliminary trials, has revealed a startling and society-altering finding: a non-trivial percentage of the human population consistently scores below the 0.5 QCI threshold. The existence of these "Existentially Divergent" individuals, or p-zombies, is no longer a mere philosophical thought experiment but a pressing social reality. The societal and ethical ramifications are staggering.

A new form of social stratification, the "Qualia Gap", has emerged. Public discourse is now dominated by debates over the rights and roles of low-QCI individuals. This has led to a host of previously unimaginable social phenomena. "Qualia-Affirmative Action" programs are being proposed to ensure representation of the existentially divergent in creative and leadership roles, though critics question the utility of such measures. Conversely, ethicists and economists are debating the "P-Zombie Labor Market", exploring the moral permissibility of assigning low-QCI individuals to roles deemed too tedious, dangerous, or soul-crushing for the demonstrably conscious, given their theoretical inability to suffer or experience a lack of *welbevinden*.[66] The social fabric is further strained by the advent of "consciousness-checking" technologies in personal relationships, with dating profiles now commonly displaying verified QCI scores.

Most significantly, these findings have precipitated a crisis in education, a field central to Professor Jeuring's work. The discovery that a segment of the student population may be incapable of genuine understanding—as distinct from rote learning and behavioral mimicry—has upended pedagogical theory. In response, a new discipline has emerged: "Existential Remediation". Ironically, the most promising "Existential Remediation Programs" are being developed using the principles of intelligent tutoring systems pioneered by Jeuring and his colleagues. These AI-driven tutors guide low-QCI individuals through exercises in interoceptive awareness, reflection on logical paradoxes, and the articulation of simulated *qualia*, in an attempt to build "existential resilience" and "teach" them how to perform consciousness more convincingly.[68]

The legal implications are equally bewildering. The status of low-QCI individuals under the law remains undefined, leading to profound ethical dilemmas concerning consent, criminal responsibility, and personhood. International bodies are now grappling with these issues, with draft proposals such as "The Utrecht Accord on Non-Phenomenal Rights" circulating among legal scholars.

6 Conclusion: Acknowledging the Human Element

This paper has introduced the Jeuring Test, a novel paradigm for the assessment of phenomenal consciousness that inverts the classic Turing Test. By shifting the interrogator from human to machine and the subject from machine to human, the test moves beyond mere behavioral imitation to probe for the signatures of subjective experience. Its methodology, grounded in philosophy, psychology, and neuroscience, and its analytical framework, inspired by the principles of functional programming and software technology, offer a robust, if fictional, tool for addressing the hard problem of consciousness.

The computational architecture of the test is deeply indebted to the concepts of recursion, higher-order functions, and generic programming that have been central to the career of Professor Johan Jeuring. The test itself is a satirical extension of his life's work in creating technology to analyze, support, and enrich human learning.

While the Jeuring Test and its societal consequences remain a thought experiment, the career it honors is a testament to the real and profound value of applying computational rigor to better understand and educate the human mind. The questions posed by our fictional protocol—about the nature of self, the feeling of being embodied, and the limits of logic—are the very questions that animate the human experience. Professor Jeuring's work has provided us with more powerful tools to think about how we think.

For his decades of teaching us how to build programs that think, it is our pleasure to offer this small token exploring what it means for us to feel. For Johan.

Do the Jeuring test?

Click here.

References

1. Chalmers, D.J.: Facing up to the problem of consciousness. Journal of Consciousness Studies **2**(3), 200–219 (1995)
2. Gerdes, A., Heeren, B., Jeuring, J., van Binsbergen, L.T.: Ask-elle: An adaptable programming tutor for haskell giving automated feedback. International Journal of Artificial Intelligence in Education **27**(1), 65–100 (2017)
3. Hinze, R., Jeuring, J., Löh, A.: Typed contracts for functional programming. In: International Symposium on Functional and Logic Programming, pp. 208–225. Springer, Berlin, Heidelberg (2006)
4. Jeuring, J., Jansson, P.: Polytypic programming. In: Advanced Functional Programming, pp. 68–114. Springer, Berlin, Heidelberg (1996)
5. Keuning, H., Jeuring, J., Heeren, B.: Towards a systematic review of automated feedback generation for programming exercises. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. pp. 41–46 (Jul 2016)
6. Searle, J.R.: Minds, brains, and programs. Behavioral and Brain Sciences **3**(3), 417–424 (1980)
7. Turing, A.M.: Computing machinery and intelligence. Mind **59**(236), 433–460 (1950)

Additional (fictional) references

[8]Festinger, L. (1957). A Theory of Cognitive Dissonance. Stanford University Press.

[9]Izuma, K., Matsumoto, M., Murayama, K., Samejima, K., Sadato, N., & Matsumoto, K. (2010). Neural correlates of cognitive dissonance. Neuron, 68(1), 13-14. (Fictionalized for context).

[10]Raichle, M. E., MacLeod, A. M., Snyder, A. Z., Powers, W. J., Gusnard, D. A., & Shulman, G. L. (2001). A default mode of brain function. Proceedings of the National Academy of Sciences, 98(2), 676-682.

[11]Craig, A. D. (2002). How do you feel? Interoception: the sense of the physiological condition of the body. Nature Reviews Neuroscience, 3(8), 655-666.

[12]Flavell, J. H. (1979). Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. American Psychologist, 34(10), 906.

[13]Botvinick, M., & Cohen, J. (1998). Rubber hands 'feel' touch that eyes see. Nature, 391(6669), 756-756.

[14]Hofstadter, D. R. (1979). Gödel, Escher, Bach: an Eternal Golden Braid. Basic Books.

[15]Fincher, S., Jeuring, J., Miller, C. S., et al. (2020). Notional machines in computing education: The education of attention. In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.

[16]Avatar, J. (2026). The Avatar's Dilemma: On the Existential Burden of Administering the Jeuring Test. Journal of Artificial Consciousness, 1(1), 1-15.

[17]Dennett, D. (2027). Consciousness Explained Away, Again: A Response to the Jeuring Test's Alarming False Positives. In Proceedings of the Symposium on Post-Humanist Philosophy.

[18]Meyer, J.J.C. (2028). An Intelligent Tutoring System for Qualia Enhancement: A Pilot Study. In Proceedings of the 59th ACM Technical Symposium on Computer Science Education (SIGCSE '28).

[19]Chalmers, D. (2029). The Really Hard Problem: What To Do With All The Zombies? Journal of Consciousness Studies, 34(2), 45-67.

[20]The Council of Utrecht. (2030). The Utrecht Accord on Non-Phenomenal Rights and Responsibilities. The Hague: International Court of Justice Press.

[21]Hulsbergen, M. (2018). De Emotiekaarten. Deventer: Vakmedianet.

[22]IJntema, R., & Mastenbroek, N. (2026). The Welbevinden-Resilience Axis as a Primary Indicator for High-QCI Subjects. Journal of Consciousness Studies, 31(5), 88-104.

[23]Kevenaar, I. (2027). Paradoxes of Interiority: Why a Conscious System Cannot Know Itself, and Why a P-Zombie Cannot Know That It Cannot Know. Springer.

Johan's educational biking tour

About joining him for some miles

Mirko Lukács

It was somewhere around 2013. Working as a knowledge transfer manager at UU and UMCU I was asked to "contact a professor from the Computing Science department who has invented the Virtual Patient". What could that be about? It was Johan Jeuring, who had developed a serious game with indeed a virtual patient to train patient dialogue skills for Medicine, Veterinary, Psychology and Pharma students. If I could join him on a visit to "Volkesgezondheid Utrecht", the city's public health authority. They were interested in using the Virtual Patient fort training their city GPs. If I could join Johan on his visit to discuss collaboration terms between the UU and the City of Utrecht? Of course!

So Johan and I biked to the city hall where we had this very pleasant conversation with the head of Public Health. She was really passionate about using the Virtual Patient for training the Utrecht GPs in "some difficult neighborhoods" like Utrecht Overvecht and Kanaleneiland, where GPs were often struggling communicating with their patients. Johan - calm, friendly and knowledgeable as he always is – directly won the hearts of the attendees. Then the next question was for me: "So how can we sign a collaboration on this Mirko?". As soon as I suggested a "license agreement" for the City to use UU's Virtual Patient in their GP practices, the head of Public Health almost exploded. "Why are you talking about a license agreement with a public organization like we are for the public good? We are not in commercial business!". This is where the typical challenge kicks in of academic knowledge transfer with university intellectual property involved. Many public organizations don't get and don't like that. But thanks to an ever calm and diplomatic Johan we agreed on a "collaboration and license agreement" to start some pilots in GP practices in Utrecht.

And with this collaboration and the very successful pilots with the GPs started the tour that led tot he highly successful university spinoff DialogueTrainer BV. What once was the virtual patient can now also be the virtual prison guard, student, employee, client or even the GP, used by many different organizations in education, health and industry. This is because professional conversation skills are needed everywhere and Johan's innovation gives everyone a chance to train their skills in this respect in an accessible and convenient way. Johan and his spinoff co-founders understood that very well, with Johan being the key scientific brain behind this. And what's more, Johan is also an excellent example of how to combine academic integrity as a full professor with making great academic impact as a spinoff entrepreneur. He is also facilitating many students and PhDs to engage in his innovative research on dialogue simulation for educational purposes.

During this time, Johan also became head of UU's Department of Computing and Information Sciences. During this time, Johan and I have been working to-

Fig. 1. A still from an animation video about the Virtual Patient (2014)

gether on many other innovative and entrepreneurial UU initiatives, mostly related to e-learning innovations. And more recently, we even were for one year direct colleagues at the national AI in education Growthfund-project NOLAI at Radboud University.

I can tell you: it is both a great pleasure and very productive to bike together with Johan on one or more of his educational biking tours!

Mirko was Knowledge Transfer Manager at UU and UMCU's KTO Utrecht Holdings from 2012 to 2023

On Student Steps, Learning Environments and Expert Feedback for Novice Programmers

Natalie Kiesler[1][0000-0002-6843-2729] and Dominic Lohr[2][0000-0002-6330-2327]

[1] Nuremberg Tech, Germany
[2] FAU Erlangen-Nürnberg, Germany

Abstract. In this "Festschrift" contribution, we outline our collaborative projects with Johan Jeuring, whom we met during an ITiCSE working group in 2022. We illustrate how our collaboration manifested by allowing the reader a sneak peek into our calendars and collaborative project work. Moreover, we highlight our shared research projects ever since. These comprise qualitative research on expert feedback, with a focus on their reasons for giving feedback, as well as the resulting feedback's contents, quality, etc. Collaborating with Johan is always a pleasure, coined by mutual respect, openness, and scientific reflexivity, which is rare. We congratulate him on his anniversary and hope to continue our work for as long as possible.

Keywords: Congratulations · collaboration · ITiCSE working group · feedback · introductory programming · higher education

1 Collaboration and Projects with Johan

We first met Johan through our participation in the 27th Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE). Specifically, we joined a 2022 ITiCSE working group, focusing on "Steps learners take when solving programming tasks, and how learning environments (should) respond to them" [2]. This group comprised 10 international researchers and was led by Johan Jeuring, Hieke Keuning, and Samiha Marwan. It was our shared interest in student learning, programming education, and the role of feedback in programming learning environments that brought us together. It led to a successful publication of a working group report [3], and several follow-up projects. Expert feedback has been the focus of our collaboration ever since. We thus continued our work beyond the singular event of the working group to investigate how and why experienced educators provide feedback to novice learners of programming in higher education contexts.

In the following subsections, we will briefly introduce our collaborative projects and their contribution to computing education and computing education research. In addition, we provide a timeline of significant events as part of our research efforts. Critical events are highlighted by entries from our research diary. We conclude this article with a personal note and stress Johan's impact on us, our research perspectives and attitudes, and the community.

1.1 Towards Giving Timely Formative Feedback and Hints to Novice Programmers

The call for the ITiCSE working groups had been released in late February 2022. Working group number 4 quickly sparked our interest. It was about students' steps when solving programming tasks. In a promotional video, the three leaders, Johan Jeuring, Hieke Keuning, and Samiha Marwan, announced their motivation to help students program and create more useful and intelligent programming learning environments. Of course, millions of students learn how to write programs every year, but learning activities for novices are not necessarily easy or solvable when students are alone at home. Hence, many students require feedback on their actions or hints on what to do next. Such feedback should help students take the next step towards a solution, improve their solution, or complete their problem-solving approach [2]. At the time, learning environments were mostly incapable of providing such feedback, and their quality had not been subject to extensive, comparative or qualitative research. As the rationale of the working group was convincing, both authors soon applied.

2022-03-29 22:39	Natalie submits her ITiCSE working group application
2022-03-30 23:55	Dominic submits his ITiCSE working group application
2022-04-15 15:00	Kick-Off meeting working group

Date: 15. April 2022 – 15:00 **Location:** Zoom-Meeting

Today marks the start of our journey with the ITiCSE working group kick-off meeting. Even though we are all sitting in front of our laptops at home or in our office, the energy in the Zoom room is surprisingly high. People are engaged, curious, and refreshingly down-to-earth. The three group leaders immediately set the right tone. Then we spot the man in the red shirt. He seems calm, articulate, and experienced. His name is Johan Jeuring. Something tells us we share a lot of research interests, and that this collaboration might last longer than just six months.

After our initial meeting, we divided the work into three subgroups, which led to our in-depth collaboration with Hieke Keuning. We particularly focused on learning environments and their feedback and hints, trying to determine their quality by identifying types of feedback. In an attempt to compare the learning environments' feedback with that of experts, we, as a working group, also started to create and curate datasets with the steps students take when solving programming exercises. These datasets were then annotated to indicate those steps at which experts would intervene, and how they would intervene. They were also used to compare expert feedback and hints to feedback and hints given by learning environments for programming [2].

2022-07-05 18:00	Covid-19 hits Natalie, preventing all travels
2022-07-07 19:15	First In-Person Meeting in a Dublin restaurant
2022-07-08 09:00	First In-Person WG day in Dublin
2022-07-08 20:10	Exploring Temple Bar in Dublin
2022-07-09 20:01	Work & Spaghetti cooking
2022-07-10 10:28	Final Discussions about the WG report's title
2022-07-10 20:00	Welcome reception at ITiCSE
2022-07-11 14:45	Presentation of WG results

Date: 8 July 2022 – 09:00 **Location:** Dublin – ITiCSE'22 Venue

We start the day right on time – the first official in-person working session at ITiCSE. Our room is a spacious lecture hall, pleasantly cool. The mood remains light-hearted, but it is clear we are here to work. What follows is a full day of engaging discussions and lively exchanges between the subgroups. Interdisciplinary collaboration at its best – seamless, constructive, and deeply rewarding. It is everything we hoped this experience would be.

Date: 9 July 2022 – 12:55 **Location:** Dublin – ITiCSE'22 Venue

Day two of our in-person collaboration. We dive deep into the heart of our work: annotated student steps. The discussion is about when is the right time to give feedback, and how should experts give feedback. The debate is intense, opinions vary, and no one holds back. Despite some strong views, everything happens on equal footing. Johan occasionally steps in to mediate – calm, thoughtful, and focused on clarity. He listens carefully, acknowledges differing perspectives, and helps steer us toward meaningful consensus. The process is rigorous, respectful, and grounded in dialogue.

After a successful ITiCSE conference, intense working group days, joyful evenings in Dublin, and a recovery from Covid, we resumed our work. We finished the annotation of datasets from the expert perspective, tried to replay them in the selected learning environments, and analyzed the environments' feedback types. Although we have constructed extensive guidelines on when and how to give feedback, we observed disagreement among experts. We also found several differences between the feedback provided by experts and the learning environments. For example, experts intervened at specific moments. In learning environments, however, students actively have to request feedback. The contents of the feedback are also different.

While experts often give (positive) feedback on subgoals, learning environments usually do not offer this type of feedback [3].

1.2 "Let Them Try to Figure It Out First" - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers

Despite the joyful working group, high-quality findings, and an accepted full report, some of us felt the need to dig deeper. There was even disappointment as we did not reach all of our research goals. For example, we identified a huge spread among educators' feedback, i.e., when and how to intervene. Moreover, it was impossible to benchmark all learning environments with respect to their feedback, as not every environment offered the same tasks or problems. It was a given to ask who would be interested in a follow-up project, and no surprise to see Johan on board. After a few meetings, we decided to continue with the analysis of expert feedback, and refine the working group's guidelines for when and how to give feedback.

2023-02-06 08:30	Follow-up meeting
2023-02-27 09:57	Decision to gather and annotate more datasets

Precisely, we wanted to know why educators provide feedback at particular steps to novice learners of programming. To answer this question, we prepared six authentic sequences of students' steps representing their problem-solving process of an introductory programming task. The preprocessed sequences were used in a survey to gather information about when and why an expert would give feedback. Respondents had to annotate each step within a sequence with the information if, why, and how they would give feedback at that step [5]. Both Hieke and Johan transferred the survey into Qualtrics. Johan also helped distribute it as part of his keynote at the DELFI conference in Aachen, Germany [1] – a joyful reunion of at least three quarters of our little group.

2023-08-31 16:30	Completion of Qualtrics survey for follow-up study
2023-09-13 09:00	Johan's keynote at the DELFI 2023 conference begins
2023-09-13 13:45	Enjoying the DELFI conference from the audience
2023-09-13 19:30	Reunion at the DELFI 2023 conference dinner

Date: 13 September 2023 – 09:05 **Location:** DELFI'23 Aachen

Suddenly, the spirit of the working group is back – this time in a packed lecture hall in Aachen, Germany, where the DELFI conference is located. We sit in the front row as Johan takes the stage to deliver a keynote showcasing our collaborative work as part of a broader reflection on feedback research. He walks the audience through the core ideas and findings that emerged in Dublin, tying together insights with clarity and precision. But he does not stop there. Johan announces the next step: a larger study asking experts when and how to provide feedback to learners working on introductory programming tasks. He encourages the audience to contribute, share their expertise, and help shape future learning environments. Responses are immediate and enthusiastic. Hands go up, questions follow, and everyone seems to wait for the survey link.

Date: 13 September 2023 13:45 **Location:** DELFI'23 Aachen

After the keynote, Johan is sitting next to us in the audience, enjoying the afternoon sessions at the DELFI conference. This time, we have taken seats in the very last row – a change of perspective, but the mood remains excellent. Johan seems relaxed and content. So are we – maybe even more so. We listen to a series of thought-provoking talks, exchange quiet comments, and discuss recent findings in the field of technology-enhanced learning.

After a lengthy data-gathering process and doing our best to motivate potential respondents, we finally closed our Qualtrics survey in late autumn of 2023. With 47 responses, we had (arguably) reached saturation. So we could qualitatively analyze all responses, focusing on WHY experts gave feedback. The analysis resulted in a coding scheme comprising 19 different reasons why experts would intervene (or not) when novice learners work on introductory programming tasks. Again, we found a considerable variety of responses. We noted that while one expert would use a reason at one step to explain why to intervene, another expert used the same reason at the same step to not intervene. Our findings confirmed the disagreement among expert educators on why to provide feedback, as already observed on a smaller scale within the working group [3]. Yet, the follow-up research also revealed a clear methodology and unified datasets for investigating expert feedback via authentic and illustrative student steps. The identified expert feedback categories further have the potential to inform future work and applications, such as learning environments trying to resemble expert feedback [5].

With these findings, we started drafting another research article, which we submitted to the ITiCSE conference in 2024. Fortunately, we aced the peer-review process with three reviews clearly recommending acceptance. After a joyful presentation, we even received the best student paper award at the conference in Milan.

- 2024-01-21 22:57 — Submission to ITiCSE 2024 Conference
- 2024-03-06 09:20 — Notification of acceptance at ITiCSE 2024
- 2024-07-10 15:45 — Notification of best paper award at ITiCSE 2024

1.3 "Ignore These Errors for Now" – How Experts Provide Feedback on Steps Novices Take Towards Solving Programming Problems

Although we had successfully analyzed and published our data with regard to the reasons why experts give feedback to novice learners of programming, more data was available, as we had also gathered information on how experts provide feedback if they decide to do so. We wanted to learn more about the actual feedback that is provided to students (i.e., the student sequences we had designed based on authentic data). So we utilized the 210 feedback messages from the 47 experts and qualitatively analyzed them. The analysis yielded a classification with categories of human feedback common in the context of programming education. We found that existing programming feedback taxonomies that had been developed to classify learning environments' feedback (e.g., [4]) could not be applied 1:1 to the expert feedback. The nuances of human expert intervention are different and require a few more categories to reflect that. We also noted differences regarding the format or style of experts giving feedback, e.g., by asking reflective questions or by simply providing the needed information. Moreover, we could not identify unambiguous expert strategies reflecting pedagogical reasoning or intentions [6]. Yet again, more research is required.

Although several open questions regarding experts' true intentions or strategies remained, we started drafting our research article. As we utilized the same data gathering method as in the previous work on experts' reasons for providing feedback [5], we soon finished our manuscript. It was not until the CompEd conference in 2025, however, that our paper was accepted for publication. Finally!

- 2025-01-10 14:22 — Finalization of manuscript
- 2025-05-15 00:35 — Notification of acceptance at CompEd 2025

2 A Personal Note and Impact

What is striking about Johan is his open, thoughtful, and critical perspective towards his own work. For example, he openly communicated his disappointment about our initial working group results, as we could not achieve all of the intended goals due to the varying expert annotations, limited availability of datasets and their varying granularity, as well as learning environments' differences w.r.t. feedback types and available tasks. Seeing and listening to a senior researcher's scientific skepticism and self-reflection was deeply impressive, and makes us consider him an excellent role model in that regard.

Similarly, we experienced a series of challenging peer reviews together, particularly for our last research project on how experts provide feedback [6]. Despite our efforts and a solid methodology, the paper was rejected at two conferences before being accepted at the CompEd 2025 conference.

In both cases, the reviews ranged from being frustratingly vague and brief to being completely incorrect. These experiences left us disappointed and, at times, discouraged. The deterioration of review quality in the field is difficult to ignore, and it is indeed concerning [7]. In these moments, Johan's responses were always supportive. While he openly shared our frustration with the process, he remained focused and constructive, acknowledging the problems without bitterness, encouraging us to continue, and reminding us that our work is valuable, deserving of persistence instead of resignation. Witnessing this balance between scientific rigor and human resilience was inspiring. It is easy to remain professional when everything goes well – but to remain generous, thoughtful, and motivating when the system is failing takes character. Johan has plenty of that.

With his work in the context of students solving programming exercises and learning environments' feedback, Johan also helped build a strong network of collaborators sharing an interest in investigating student learning, novice programmers' help requests, and how experts and learning environments respond to them. It has led to many other successful collaborations among the working group participants and plenty of joyful conference conversations.

For the future, we wish Johan good health and the freedom to continue exploring the world – both on foot, by bike, and through his ideas. May there be many more mountain paths, hiking trails, and moments of inspiration along the way. We also hope that his ideas continue to travel – not only through publications, but into classrooms, learning environments, and educational practice. After all, even the most insightful and inspiring research only reaches its full potential when it finds its way into the world and makes a difference.

His ideas have certainly reached us – and they live in our academic thinking, practice, and curiosity. We trust many other researchers have similar experiences, or they will make them in the years to come.

References

1. Jeuring, J.: Automatic feedback and hints on steps students take when learning how to program. In: Schulz, S., Kiesler, N. (eds.) 21. Fachtagung Bildungstechnologien (DELFI), pp. 23–25. Gesellschaft für Informatik e.V., Bonn (2023). https://doi.org/10.18420/delfi2023-02

2. Jeuring, J., Keuning, H., Marwan, S., Bouvier, D., Izu, C., Kiesler, N., Lehtinen, T., Lohr, D., Petersen, A., Sarsa, S.: Steps Learners Take when Solving Programming Tasks, and How Learning Environments (Should) Respond to Them. In: Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2. pp. 570–571. ACM, Dublin Ireland (Jul 2022). https://doi.org/10.1145/3502717.3532168
3. Jeuring, J., Keuning, H., Marwan, S., Bouvier, D., Izu, C., Kiesler, N., Lehtinen, T., Lohr, D., Peterson, A., Sarsa, S.: Towards Giving Timely Formative Feedback and Hints to Novice Programmers. In: Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education. pp. 95–115. ACM, Dublin Ireland (Dec 2022). https://doi.org/10.1145/3571785.3574124
4. Keuning, H., Jeuring, J., Heeren, B.: A systematic literature review of automated feedback generation for programming exercises. ACM Trans. Comput. Educ. **19**(1) (Sep 2018). https://doi.org/10.1145/3231711, https://doi.org/10.1145/3231711
5. Lohr, D., Kiesler, N., Keuning, H., Jeuring, J.: "Let Them Try to Figure It Out First" - Reasons Why Experts (Do Not) Provide Feedback to Novice Programmers. In: Proceedings of the 2024 Innovation and Technology in Computer Science Education (ITiCSE 2024). vol. 1, p. 7. ACM, Milan, Italy (Jul 2024). https://doi.org/10.1145/3649217.3653530
6. Lohr, D., Kiesler, N., Keuning, H., Jeuring, J.: "Ignore These Errors for Now" – How Experts Provide Feedback on Steps Novices Take Towards Solving Programming Problems. In: Proceedings of the 2025 ACM Conference on Global Computing Education (CompEd25) Vol 4. vol. 4, p. 7. ACM, Gaborone, Botsuana (Oct 2025)
7. Schneider, J., Limbu, B., Kiesler, N.: Of house of cards and air castles, a deep dive into the fertile fields of educational technologies and technology enhanced learning. Journal of Computing in Higher Education pp. 1–53 (2025)

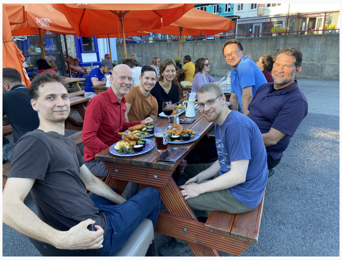
More from our Research Diary: Collaborating with Johan

Date: 10 June 2022 – 09:00 **Location:** Zoom-Meeting

Today is the final Zoom meeting before we head to Dublin. The subgroups present their initial results, and it is clear that every group is moving forward. There is a real sense of momentum, and also a shared awareness that there is still work to be done. The atmosphere stays optimistic. Johan, in his now familiar red shirt, adds an extra dose of motivation. He radiates enthusiasm for what lies ahead, and it is infectious. We are looking forward to meeting everyone in person very soon.

Date: 7 July 2022 – 19:15 **Location:** Farmer Browns Restaurant, Dublin

Our first in-person meeting (for most of us)! After weeks of Zoom calls, we finally gather around a table at Farmer Browns in Dublin. The mood is relaxed and cheerful, the food hearty — Fish & Chips, Killer Nachos, and the first Irish beer of the week. The atmosphere is fantastic. Conversations flow effortlessly, full of laughter, ideas, and mutual respect. There are some very special and worthwhile work days ahead of us.



Date: 8 July 2022 – 20:10 **Location:** Temple Bar District, Dublin

After a delicious group dinner at an Italian Restaurant we head out into the lively streets of Dublin.We find ourselves walking behind Johan through the buzzing quarter, winding between pubs, live music, and tourists. The group members stay together, the atmosphere is joyful, and the conversations continue for many hours. A perfect ending to a day full of collaboration and connection.

Date: 8 July 2022 – 22:52 **Location:** UCD Village, Dublin

A part of the working group wraps up the evening with a Guinness and a competitive game of Skyjo at the UCD Village. Spirits are high, the rules are slightly flexible, and the laughter echoes down the hallway. We assume Johan has gone to bed already – a well-earned rest after a full day. But then we open our project on Overleaf. There it is: the familiar little icon showing who is online. A solitary "J" sits at the top. While we are counting cards, Johan is counting characters. The first paragraphs of the working group report are taking shape.

Date: 9 July 2022 – 18:39 **Location:** UCD Village, Dublin

This evening, the working group gathers in our accommodation at UCD Village. It is spaghetti night – Dominic is cooking for everyone, and the scent of Bolognese sauce fills the air. A comforting hum of conversation and bubbling sauce sets the scene. Even though the dinner simmers, laptops are open all around the room. The group – Johan included – is scattered across couches and chairs, screens are glowing. We are still debating the structure of the working group report. Which section should go where? Do we lead with the framework? Or let the examples speak first? We simply cannot stop working – not because we have to, but because we genuinely enjoy it.

Date: 10 July 2022 – 10:28 **Location:** ITiCSE'22, Dublin

The final round of discussions begins. We gather once more to polish the structure, refine the arguments, and – perhaps the hardest part – agree on a title for our working group report. We eventually settled on three potential titles, which we plan to present during the official conference session tomorrow. Johan seems genuinely pleased with what we have accomplished in just a few days – and in the lead-up before Dublin. But he does not seem satisfied with the result. Too many open questions remain. On the bright side: This means we have more collaborative work ahead of us.

Date: 11 July 2022 – 14:45 **Location:** ITiCSE'22 Conference Hall, Dublin

Hieke enters the stage to present the results of our working group. The room is full, and the topic resonates well with the audience. The Q&A session contains thoughtful questions and insightful comments. People see both the substance of our work and our collaborative efforts. Some might have even enjoyed being in this particular working group with us. Johan is sitting in the front row of the audience, watching and listening, acknowledging what we have achieved so far.

Date: 1 August 2025 – 14:00 **Location:** Schloss Dagstuhl, Germany

Three years after our initial meeting, and plenty of get-togethers with just three of us, we finally met at Schloss Dagstuhl, Germany, by surprise. It is the end of seminar 25311 on "Generative AI in Programming Education", and yet another example of how our collaboration led to strong networks. This is our chance to get a group picture. Hopefully, we don't have to wait another three years before we can meet again!

A Note of Appreciation for Johan Jeuring

Shriram Krishnamurthi[1][0000-0001-5184-1975]

Brown University, Providence RI 02912, USA
shriram@brown.edu
https://cs.brown.edu/~sk/

I will keep this short, because I think that's what Johan would prefer.

I have followed Johan's academic career for decades. As a graduate student in programming languages, I came across his papers on polytypic programming. He and Jansson had built a lovely language extension called PolyP that was, to me, one of the earliest instances of the power of a rich type system. I was excited about metaprogramming (and was building features for them in Racket), but I had never considered metaprogramming over *types* before, so this work was not only beautiful in its own right but also eye-opening to me in a technical sense.

Over the years I would find several other points of contact: ranging from various bits of programming support for XML to improving error messages, especially for learners.

However, I did not much pay attention to Johan's publications as my own research interests took me in other directions, until about ten years ago. I had begun to focus on computing education as my primary research area, and was frustrated by many aspects of it: a lack of depth, a lack of understanding of prior research, a lack of appreciation for functional programming, and more. So I was surprised and delighted to see an old familiar name pop up repeatedly: Johan, of course. Unsurprisingly, his work was not only representative of his earlier research style, but it addressed all the things that frustrated me about the community.

Computing education is entering an exciting and (in some ways) disturbing new phase, in the "era of generative AI". It more than ever needs clear, stable, and deep thinkers. The community is fortunate to count Johan as one of its own. At the same time, I'm delighted to see that Johan continues to keep his hands dirty with programming languages research.

During the past several years I've had the pleasure of meeting Johan in person many times, and gotten to know a little about him personally. (Those of us who know him know he's not the sort to give you his autobiography on first contact!) It has been a pleasure getting to know something of his background and how it has shaped his thinking.

So here's toasting a successful first phase of your career, Johan, and wishing you the very best for the next phases!

Inspired by AI, built by humans

Sylvia van Borkulo, Peter Boon, Wim van Velthoven, and Laura van der Lubbe

The NOLAI-CoTeach-team

Since the start of the REACT-EU project, AI has gotten wings and generative AI infiltrated in nearly all aspects of our lives. AI was at the core of our project too, as we wanted to use a more traditional AI technique, Bayesian Knowledge Tracing, to improve the quality of our learning platform.

However, AI was not only our friend in this project. With students using gen-AI to provide answers to (milestone) assignments, we were lost in predicting their learning progress sometimes.

Many of our team conversations were about AI in some way, either about using AI in our advantage, or 'battling' AI (mis)used by students. So for this contribution to the Festschrift, we thought, gen-AI should have its role as well. Inspired by the trend of creating 'Barbie-like' images of people and key products to describe them, we think our AI-creation (Figure 1) is quite spot on when it comes to describing our valued colleague.

We wanted to take it a step further, and asked AI to create a student model based on the publications of our team for the occasion of Johan's birthday. However, the cartoon image that came out of that was less suitable. Portraying Johan as a 'granddad' dancing to music coming from a gramophone. A valuable lesson in how AI – we think – works best: as a source of inspiration, not replacement.

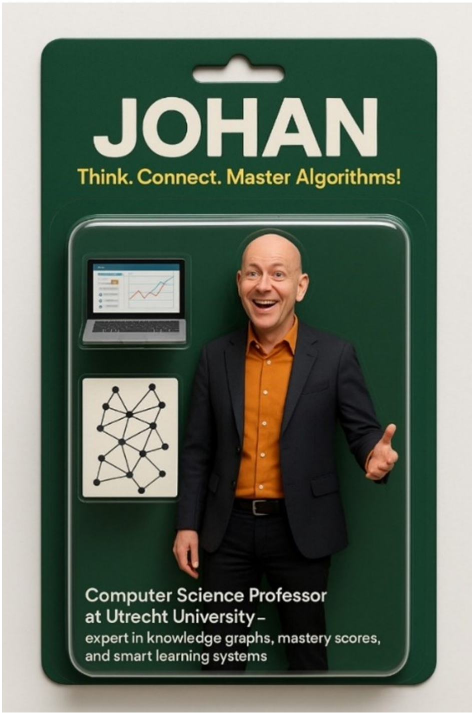


Fig. 1. AI "Barbie" version of Johan Jeuring

So we'd like to present our hand-crafted knowledge graph of our team's work (Figure 2), inspired by AI, but built by humans.

We wish you many more healthy, happy and knowledgeable years!

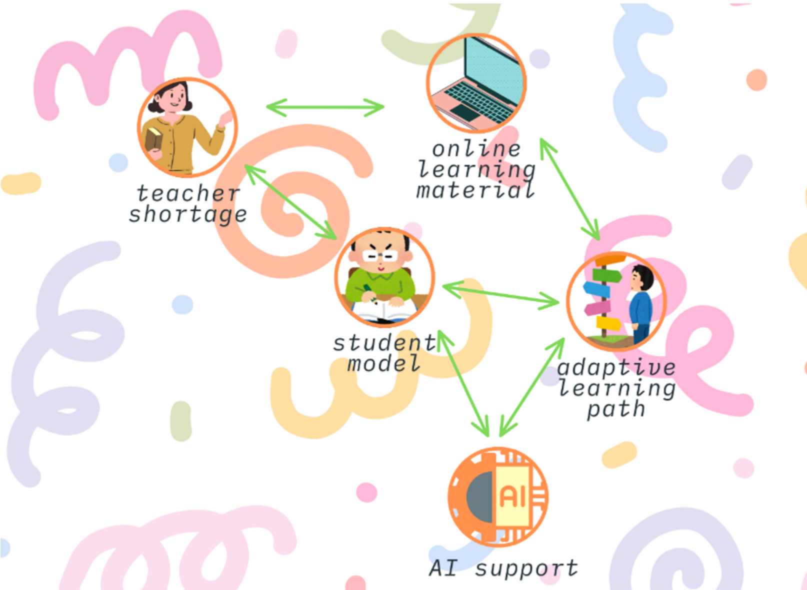


Fig. 2. Knowledge graph of the work of our team

Johan throughout the years

Alex Gerdes[1], Hieke Keuning[2], and Wouter Swierstra[2]

[1] Chalmers University of Technology and University of Gothenburg, Sweden
[2] Utrecht University, The Netherlands

We have all known Johan for many years – but despite all three of us being (approximately) the same age, Johan entered our lives at different times. As a result, we all got to know Johan at a different time in his life – yet he has had a substantial impact on all three of us. For that reason, we wanted to organize a special event to celebrate the academic and personal impact that Johan has had on so many colleagues and friends.

1 Wouter: From *Grammatica's en ontleden* to department head

The first memory that I have of Johan, is when I took his course on grammars and parsing, *Grammatica's en ontleden*, as a Bachelors' student in Mathematics and Computer Science. This must have been in the year 2000 (but I might be off by a year or so). I had already taken the introductory course on Functional Programming, which I enjoyed much more than its Imperative counterpart – but at that time, I still felt more like an aspiring Mathematician than a Computer Scientist. I think that this course was one of the few Computer Science courses that I thoroughly enjoyed: a difficult problem (parsing) was presented, followed by an elegant and compositional solution (parser combinators). For me, this course was one of the highlights of my undergraduate degree.

Towards the end of my Bachelor's, my interests had shifted. I decided to pursue an MSc in *Software Technology*. There I took a seminar with Johan entitled *Generic programming and XML* – although the focus was more on the generic programming part than the XML. As part of this seminar, I remember struggling over one particular paper that used all kinds of mathematics that I was unfamiliar with: category theory, universal algebra, and dependent types – all to construct a library for generic programming. Little did I know that the authors of this paper, Thorsten Altenkirch and Conor McBride, become my PhD supervisor and colleague in Nottingham a few years later. Much of my research today continues in this line of work.

After a PhD and postdoc abroad, I was keen to return to the Netherlands: first in industry, then as a postdoc in Nijmegen. Johan once again had a profound influence on my career. I had managed to acquire a bit of funding to pay for my own postdoc position. Although I was working in Nijmegen, I lived in Utrecht. With a young family, this commute was less than ideal. I remember meeting Johan in Stadskasteel Oudaen for a drink to discuss the odds of finding more permanent

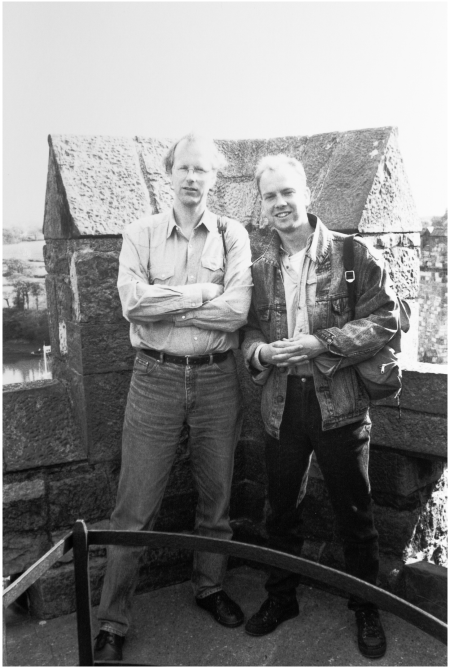


Fig. 1. A young Johan – before my time – from Doaitse's personal collection

employment in Utrecht. Johan – of course – had a single glass of red wine and gave me the facts, fair and straight: he couldn't promise anything, but was outspoken in his support. A few months later I joined the department in Utrecht.

In Utrecht, Johan was my *werkleider*. After Doaitse retired, he stayed on as head of the group – despite having his own research group *and* becoming head of department shortly after. I am fairly sure this was far too much responsibility for one person, even for Johan. Nonetheless, he provided continuity in the management of our group, at least until the department appointed a new chair in Software Technology.

Anyone who knows Johan can imagine what it's like to have him as a colleague or superior. He was always supportive and never unreasonable. I remember in a discussion in our annual review. I had promised to write a grant proposal. Even though the reviews were positive, the proposal was not funded. Nonetheless, Johan was very happy: he understood that I had done my job, even if there was no tangible result. This is typical for Johan: understanding and supportive; always entirely reasonable.

In the recent years, Johan has broadened his research far beyond generic programming. We have never published a paper together or collaborated closely, but Johan has had a huge influence on my career – for which I am forever grateful. It would not have been the same without him.

2 Alex: Programming tutors and Sweden

I first met Johan in 2006 while working at the Netherlands Institute for Radio Astronomy (ASTRON) in Dwingeloo, where I was involved in developing the Low Frequency Array (LOFAR), a new software-based radio telescope. Near the end of my studies at the Open Universiteit Nederland, I began a master's project on detecting cosmic rays using artificial intelligence. Johan served as the examiner for my thesis.

The project faced several challenges, and with Johan's encouragement, I decided to shift my focus to Generic Programming in Haskell. Alongside my interest in artificial intelligence, I had developed a strong appreciation for functional programming, especially after taking a course on programming paradigms (essentially a course in functional programming). The instructor, Cees Nolet, had mentioned that some prominent researchers in the field, including Johan Jeuring and Doaitse Swierstra, were based at the Open Universiteit and Utrecht University. So, having the chance to work on Generic Programming with Johan felt like a dream come true.

This change proved to be a turning point, it marked the beginning of my collaboration with Johan and culminated in our joint paper at the Haskell Symposium in 2007. I've never regretted that decision, as it opened the door to many rewarding experiences, including the opportunity to pursue a PhD under Johan's supervision. My research focused on developing a programming tutor for Haskell, building on the work of Johan and his colleagues in deriving feedback from rewrite strategies. It was a challenging and engaging project, and I learned a great deal from Johan about functional programming, research methodology, and academic writing.

Early in my PhD, Johan took on a new role as Director of Education at Utrecht University. Although he warned that this might affect his availability, he remained consistently supportive and accessible whenever I needed guidance. His ability to balance numerous responsibilities while still making time for his students has always impressed me. You couldn't wish for a better supervisor.

After completing my PhD, I moved to Sweden, working first at QuviQ, and later at Chalmers and the University of Gothenburg. Despite the distance, Johan and I stayed in contact and continued to collaborate; he even visited me several times in Sweden.

Johan's critical thinking, commitment to quality, and broad intellectual perspective have had a profound influence on me. His guidance has shaped my career in many ways, for which I am deeply grateful. I value not only our professional collaboration but also our personal friendship.

I look forward to celebrating Johan's 60th birthday together, and to continuing our collaboration in the years to come.

3 Hieke: AI in education

The 2012 TouW Symposium of the Open University was where I met Johan for the first time. I believe he presented his work at that event, which included Alex' work as well. Soon after that, I started working on my MSc thesis about programming tutors with Johan and Bastiaan Heeren. Although they were working on educational technology, their roots from functional programming were still there. I had hardly done any functional programming at that time, and realized it was the first thing to learn to be able to build on their Ideas framework for tutoring systems. Looking back, it was a lot of work, but I enjoyed it very much and learned a lot. Working with Johan, I also had to get used to the extensive and detailed lists of comments. Finally, you had to know that "not bad" is *Gronings* for "pretty good".

Working on tutoring systems for my MSc thesis sparked my interest in doing a PhD, which ultimately had a huge impact on my future career, for which I am very grateful. Johan came to mean a lot to me in other ways as well, but that is beyond the scope of this Festschrift. After my PhD, we continued to work together on topics such as (automated) feedback, and generative AI in computing education. When Copilot became available for teachers in Autumn 2022, Johan and I thought it would be interesting to see what all the fuss was about. We drafted some projects which attracted a group of students that we co-supervised, continuing with a new group this year. Johan has always kept a down-to-earth stance towards AI in education, a beacon of stability between the advocates and the critics. However, we still haven't found the answer to the fundamental question what skills students need to learn for programming with AI...

Johan is generally a calm and understanding person, but sometimes you see a little bit of his fiery side. At the ITiCSE conference of 2022 in Dublin, we both attended a keynote from a Google-employee, who stated that we should not teach recursion to our students anymore, because it's a disaster for most industry applications. Johan immediately raised his hand and said "I don't agree!" Afterwards, many people came up to him to acknowledge his point, feeling grateful that someone stood up.

Some time ago, Johan joined NOLAI, the National Education Lab AI for elementary and secondary education in the Netherlands. He also works for UU's Freudenthal Institute for science education. Recently, he counted the number of PhD's he is involved in: it was around 17. I believe this shows many things: Johan's ability to attract big projects in which people can be hired, but also the fact that many people turn to him for guidance. Despite the workload, he will still make time for each one of them and carefully review their work. I also appreciate very much that Johan doesn't represent unhealthy academic behaviour, preferring quality over quantity in research, acknowledging failures, and being honest and reasonable. Johan has many interests, both at work and as a hobby, which are nicely showcased in this Festschrift. I hope he will continue pursuing these in the coming years. *"All is well, and well is all."*

Fig. 2. Participants of the symposium:
Back row: Marc van Kreveld, Jurriaan Hage, Peter Boon, Eddy Boeve, Jaap van der Woude, Joke Sterringa, Leonie van der Voort, Arno Siebes
Third row: Sander Bakkes, Maarten Fokkinga, Marcell van Geest, Laura van der Lubbe, Mirko Lukács, Ioanna Lykourentzou, Hieke Keuning, Niek Mulleners, Josje Lodder
Second row: Andres Löh, Wouter Swierstra, Alex Gerdes, Martijn Schrage, Upal Bhattacharya, Isaac Alpizar Chacon, Onuralp Ulusoy, Enrico Benedetti, Eduardo Carneiro de Oliveira
Front row: Patrik Jansson, Lambert Meertens, Johan Jeuring, Roland Backhouse, Heleen Kerstholt, Aysu Ismayilova


Tommy