Reviewing a technical document

DAT315

The Computer Scientist in Society

1977 ACM Turing Award Lecture

The 1977 ACM Turing Award was presented to John Backus at the ACM Annual Conference in Seattle, October 17. In introducing the recipient, Jean E. Sammet, Chairman of the Awards Committee, made the following comments and read a portion of the final citation. The full announcement is in the September 1977 issue of *Communications*, page 681.

"Probably there is nobody in the room who has not heard of Fortran and most of you have probably used it at least once, or at least looked over the shoulder of someone who was writing a Fortran program. There are probably almost as many people who have heard the letters BNF but don't necessarily know what they stand for. Well, the B is for Backus, and the other letters are explained in the formal citation. These two contributions, in my opinion, are among the half dozen most important technical contributions to the computer field and both were made by John Backus (which in the Fortran case also involved some colleagues). It is for these contributions that he is receiving this year's Turing award.

The short form of his citation is for 'profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on Fortran, and for seminal publication of formal procedures for the specifications of programming languages.'

The most significant part of the full citation is as follows:

'. . . Backus headed a small IBM group in New York City
during the early 1950s. The earliest product of this group's
efforts was a high-level language for scientific and technical co

putations called Fortran. This same group designed the first system to translate Fortran programs into machine language. They employed novel optimizing techniques to generate fast
machine-language programs. Many other compilers for th
guage were developed, first on IBM machines, and late
ally every make of computer. Fortran was ado
national standard in 1966.

During the latter part of the 1950s,
international committees which develo
version, Algol 60. The language Al
pilers, received broad acceptanc
veloping programs and as a
algorithms on which the pr

In 1959, Backus pr
ence in Paris on the
national algebraic
employ a forma
ming langua
standing f
recogni

ic
tical
ges and
the most
Almost all
some type of

Can Programming P
von
Neumann Style? A
d Its
a of Prog

programming languages are growing
nous, but not stronger. Inherent defects
asic level cause them to be both fat and
primitive word-at-a-time style of program-
ited from their common ancestor—the von
computer, their close coupling of semantics to
ansitions, their division of programming into a
of expressions and a world of statements, their
ity to effectively use powerful combining forms for
ding new programs from existing ones, and their lack
useful mathematical properties for reasoning about
programs.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

General pe
or part of this ma
libraries acting for
and that reference i
to the fact that reprin
Association for Compu
table, other substantial
permission as does republ
tion.

Author's address: 91 Sa
, CA
94114.

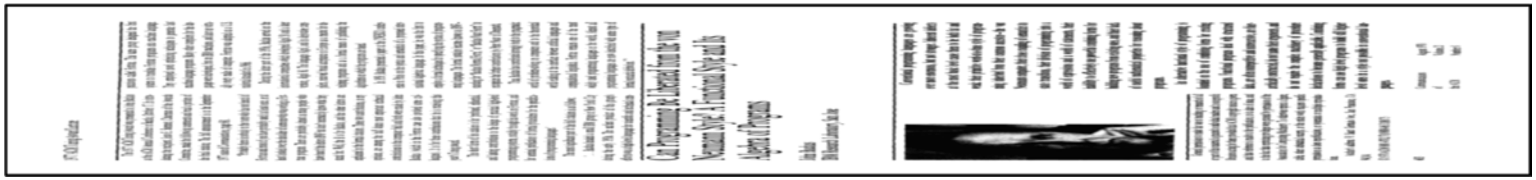
© 1978 ACM 0001-0782/78/0800

613

Communications
of
the ACM

August 1978
Volume 21
Number 8

review



Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose

- Feedback is invaluable!
- Much easier to see faults in *other people’s* writing
- *Sharpens* your own writing skills

“This paper gives wrong solutions to trivial problems. The basic error, however, is not new.”

Mathematical Reviews **12**, p. 561. Clifford Truesdell.
(1952?)

The best review I ever received was "Like all papers on category theory, this paper contributes absolutely nothing".

Graham Hutton

(It's now one of my most cited papers...)

”This has been done before in Emacs Lisp.”

Review of the QuickCheck paper

“I believe that the most important idea in the paper is the idea of using a finite automaton to model the infinite space of possible signatures.”

“At the end of paragraph A, I was happy, but but the time I got to sentence 3 of paragraph B, where it says that a machine register has a weight that is equal to the number of resources it consumes, I felt that I no longer understood what was going on.”

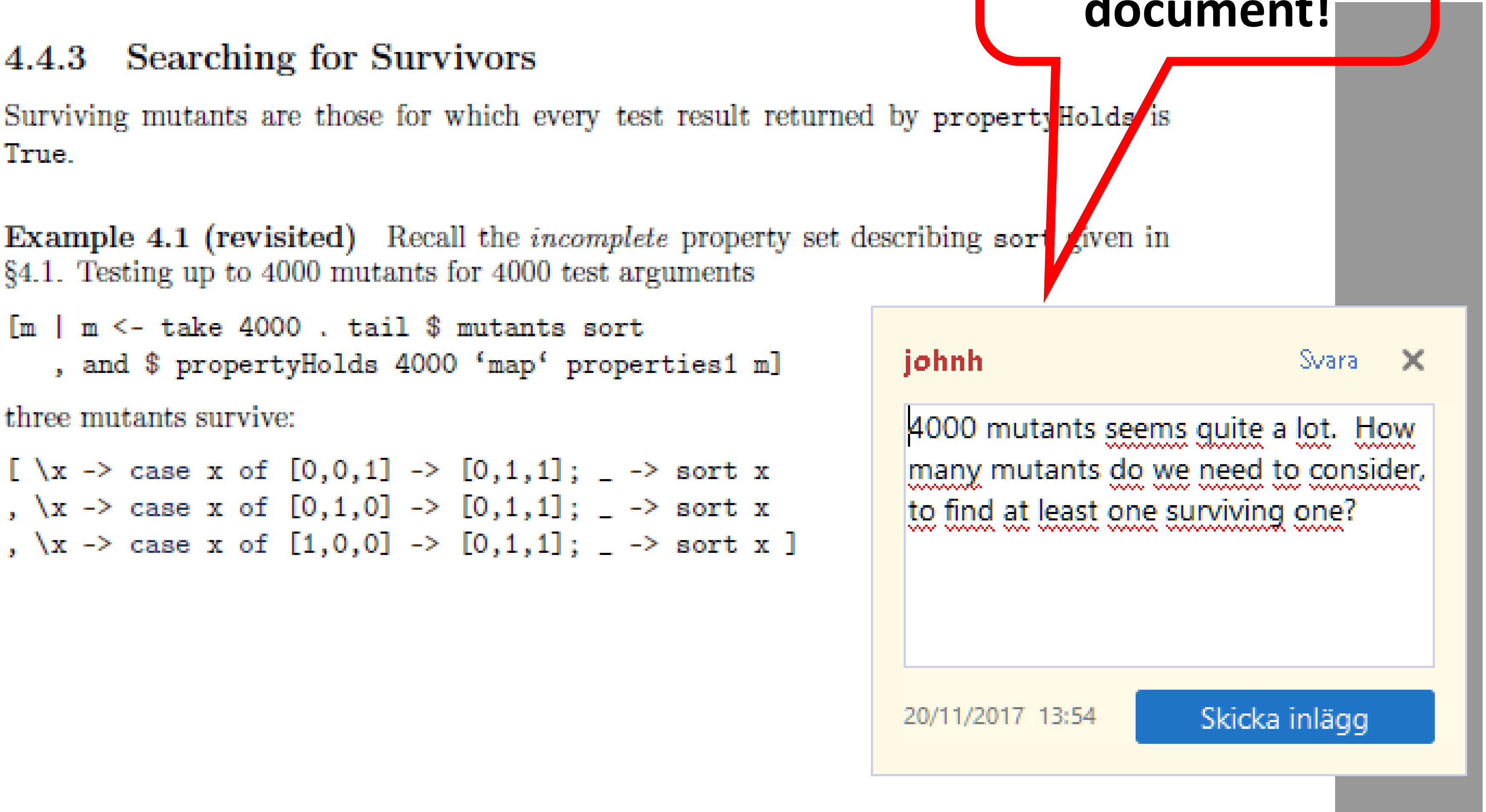
“I don’t understand the distinction between an ‘argument’ and a ‘parameter’.”

“the third section is not well written.”

How to review a document

- Read carefully

Make notes as you read, on the document!



4.4.3 Searching for Survivors

Surviving mutants are those for which every test result returned by `propertyHolds` is `True`.

Example 4.1 (revisited) Recall the *incomplete* property set describing `sort` given in §4.1. Testing up to 4000 mutants for 4000 test arguments

```
[m | m <- take 4000 . tail $ mutants sort
   , and $ propertyHolds 4000 ‘map‘ properties1 m]
```

three mutants survive:

```
[ \x -> case x of [0,0,1] -> [0,1,1]; _ -> sort x
, \x -> case x of [0,1,0] -> [0,1,1]; _ -> sort x
, \x -> case x of [1,0,0] -> [0,1,1]; _ -> sort x ]
```

johnh

4000 mutants seems quite a lot. How many mutants do we need to consider, to find at least one surviving one?

20/11/2017 13:54

What kind of notes?

- What were you thinking at each pont?
- Note your reactions.
- Note questions (even though they may be answered later)
- Mark mistakes!

Start with a summary!

This paper discusses constructs for chunked parallelisation in <…>, a functional language targetting GPUs. It explains the constructions clearly, with good motivations, in the context of three very interesting examples. It concludes with some impressive benchmarks showing very good performance. I like the paper a lot, and it is squarely in the HPC area.

Start with a summary!

Suppose, when fuzzing a protocol, we have a set of mutated messages that we want to send to the SUT, each in an appropriate state. We face the problem of getting the protocol into the desired state before sending the mutated message. We can do so by resetting the connection, navigating to the desired state by sending a sequence of unmutated messages, and then send the mutated one. The problem is that this takes time.

This paper proposes to solve this problem by continuing a test from the current state of the SUT. But after sending a mutated message, then that state is unknown! The paper proposes to figure out that state, using one of two ideas: …

Start with a summary!

This paper discusses the problems of automatically testing a DSL compiler, in the context of <…>. The problem is finding a suitable test oracle. Whereas compilers for general purpose languages can be tested against another compiler, or by comparing the result of different optimization levels, for a DSL there is normally only one compiler, and indeed, often only one optimization level. So a different approach is needed.

The paper presents three different approaches. The first…

Why summarize?

- Helps to clarify your thinking
  - Do you really understand what the document is about?
- Helps recipient see if you understood…

Give your reaction

- *“The idea is simple and easy to understand, and I am quite surprised to find that it doesn't seem to have been done before--but as far as I can tell, it is novel in this paper. That’s a clear plus.”*

- *“I found the paper well-written and very interesting; it's refreshing to see fault localization techniques applied to such an untraditional language as <…>.”*

Major vs minor comments

- Minor comments need fixing, but are easy to fix

----

p3 "As mentioned in Section ??" -- which section?

p4 "Testing the abnormal input is time time-consuming because of the infinite input space that spends more time in the state transition."

OK, this sentence I don't understand.

p8 Figure 9: you show the average time to provoke a crash on the <…>. Average of how many runs?

What questions did you have while reading?

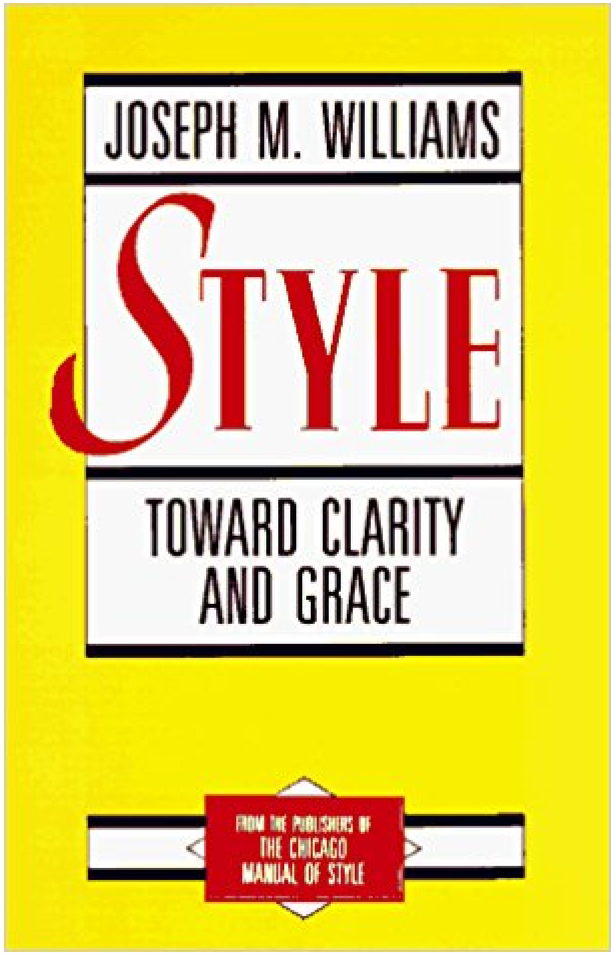
What has properness got to do with it?

“As the function properSubsetsOf returns tiers of proper sublists of values from a given tier-list, we avoid most but not all repetition.”

On page <…> you say properSubsetsOf is needed to avoid some repetition. Why does this reduce repetition? What difference does is make whether sublists are proper or not?

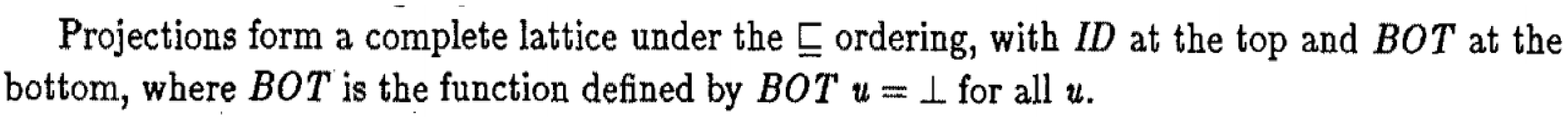
What did you find hard to understand?

I did find your discussion of Figure 1 a bit confusing. The problem is that the text discusses the farmer taking the fox across the river first, leaving the chicken to eat the grain. But when I studied the near and far states in Figure <…>, it seemed that the farmer carried the GRAIN over the river first, leaving the fox to eat the chicken! <…> I wondered: why are you calling crossRiver in the test, instead of stateTransition? Why does the call to crossRiver appear to bear no relationship to the near and far sets? What's going on?



Are Williams’ guidelines followed? If not, point it out.

Where would you like to see an example?



Projections form a complete lattice under the $\sqsubseteq$ ordering, with ID at the top and BOT at the bottom, where BOT is the function defined by $BOT\ u = \perp$ for all u.

What did you expect to see, but not find?

Was part of the argument unconvincing? Why?

On the other hand, the evaluation leaves me a little unsatisfied....the "previous approach" used for comparison is the authors' own implementation of the <…> method, rather than a real tool developed by someone else. It would have been interesting to compare performance against some of these other real tools, rather than something that could turn out to be a "straw man".

Are there parts you are sure are errors?

I do not understand how the method of paper <...>, which this paper relies on fundamentally, can possibly generate a FINITE set of timed path conditions that capture all possible information flows through a Simulink model.

Consider this: <counterexample>

- What suggestions would you make to improve the document?

  ”It might be helpful to put section 3 first…”

- Is there another paper you think the author should read?

  “see for example ASE'15 work by Cohen and Maoz and ICSE'16 work by Busany and Maoz”

Structure of a review

- Summary of the document
- Your reaction
- Major points
- Your conclusion

- Minor points

Anonymous or signed…

- Be polite, but honest in your opinion
- Realize *you* may be wrong
- Be helpful and constructive