



The Scientific Literature

John Hughes





Operating
Systems

R. Stockton Gaines
Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

Introduction

The concept of time is fundamental to our way of thinking. It is derived from the more basic concept of the order in which events occur. We say that something happened at 3:15 if it occurred *after* our clock read 3:15 and *before* it read 3:16. The concept of the temporal ordering of events pervades our thinking about systems. For example, in an airline reservation system we specify that a request for a reservation should be granted if it is made *before* the flight is filled. However, we will see that this concept must be carefully reexamined when considering events in a distributed system.

General permission to make fair use in teaching or research of all or part of this material is granted to individual readers and to nonprofit libraries acting for them provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery. To otherwise reprint a figure, table, other substantial excerpt, or the entire work requires specific permission as does republication, or systematic or multiple reproduction.

This work was supported by the Advanced Research Projects Agency of the Department of Defense and Rome Air Development Center. It was monitored by Rome Air Development Center under contract number F 30602-76-C-0094.

Author's address: Computer Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park CA 94025.
© 1978 ACM 0001-0782/78/0700-0558 \$00.75

558

A distributed system is a network of computers, each of which is connected to the others by a communication network. The system is viewed as a collection of processes, each of which is executing a program on a computer. The execution of a single machine instruction could be one event in the system.

The partial ordering of events in a distributed system is defined as follows: an event *happens before* another event if the first event is the execution of a single machine instruction on a computer, and the second event is the execution of a single machine instruction on a computer, and the first event occurs at an earlier time than the second event.

Communications
of
the ACM

July 1978
Volume 21
Number 7

The Semantic Elegance of Applicative Languages

D. A. Turner

University of Kent at Canterbury

In what does the alleged superiority of applicative languages consist? In the last analysis the answer must be in terms of the reduction in the time required to produce a correct program to solve a given problem. On reflection I decided that the best way to demonstrate this would be to take some reasonably non-trivial problem and show how, by proceeding within a certain kind of applicative language framework it was possible to develop a working solution with a fraction of the effort that would have been necessary in a conventional imperative language. The particular problem I have chosen also brings out a number of general points of interest which I shall discuss briefly afterwards.

Before proceeding it will be necessary for me to quickly outline the language framework within which we shall be working. Very briefly it can be summarised as (non-strict, higher order) recursion equations + set abstraction. Obviously what matters are the underlying semantic concepts, not the particular syntax that is used to express them, but for the sake of definiteness I shall use the notation of KRC ("Kent Recursive Calculator"), an applicative programming system implemented at the University of Kent [Turner 81]. KRC is fairly closely based on the earlier language SASL, [Turner 76], but I have added a facility for set abstraction.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1981 ACM 0-89791-060-5/81-10/0085 \$00.75

Execution of a single machine instruction could be one event in the system.

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 com-

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 the language were developed, first on IBM machines, and later on virtually every make of computer. Fortran was adopted as a U.S. national standard in 1966.

During the latter part of the 1950s, Backus served on the international committees which developed Algol 58 and a later version, Algol 60. The language Algol, and its derivative compilers, received broad acceptance in Europe as a means for developing programs and as a formal means of publishing the algorithms on which the programs are based.

In 1959, Backus presented a paper at the UNESCO conference in Paris on the syntax and semantics of a proposed international algebraic language. In this paper, he was the first to employ a formal technique for specifying the syntax of programming languages. The formal notation became known as BNF—standing for "Backus Normal Form," or "Backus Naur Form" to recognize the further contributions by Peter Naur of Denmark.

Thus, Backus has contributed strongly both to the pragmatic world of problem-solving on computers and to the theoretical world existing at the interface between artificial languages and computational linguistics. Fortran remains one of the most widely used programming languages in the world. Almost all programming languages are now described with some type of formal syntactic definition."

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

John Backus
IBM Research Laboratory, San Jose



General permission to make fair use in teaching or research of all or part of this material is granted to individual readers and to nonprofit libraries acting for them provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery. To otherwise reprint a figure, table, other substantial excerpt, or the entire work requires specific permission as does republication, or systematic or multiple reproduction.

Author's address: 91 Saint Germain Ave., San Francisco, CA 94114.
© 1978 ACM 0001-0782/78/0800-0613 \$00.75

613

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of 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.

Communications
of
the ACM

August 1978
Volume 21
Number 8

Comprehensive Comprehensions

Simon Peyton Jones, Microsoft Research

Philip Wadler, University of Edinburgh



My First Little Scientific Article

Little Old Me

Chalmers University, Göteborg, Sweden.
little.old.me@chalmers.se

Abstract

In 2009, Claessen et al. presented a way of testing for race conditions in Erlang programs, using QuickCheck to generate parallel tests, a randomizing scheduler to provoke races, and a sequential consistency condition to detect failures of atomicity [1]. That work used a small industrial prototype as the main example, showing how two race conditions could be detected and diagnosed.

In this paper, we apply the same methods to dets, a vital component of the mnesia database system, and more than an order of magnitude larger. dets is known to fail occasionally in production, making it a promising candidate for a race condition hunt. We found five race conditions with relatively little effort, two of which may account for the observed failures in production. We explain how the testing was done, present most of the QuickCheck specification used, and describe the problems we discovered and their causes.

Categories and Subject Descriptors D [1]: 1; D [1]: 3; D [2]: 5; H [2]: 4

General Terms Experimentation, Reliability, Verification

Keywords Erlang, QuickCheck

1. Introduction

In October, 2007, Torbjörn Törnkvist wrote the following on the "erlang questions" mailing list:

"We know there is a lurking bug somewhere in the dets code. We have got 'bad object' and 'premature eof' every other week the last year. We have not been able to track the bug down. The dets files is repaired automatically next time

entered at Klarna, a services to

they could be used to find race conditions in the dets code. The answer turned out to be "yes", and in this paper we describe how the testing was done, and the race conditions that we found.

2. An overview of dets

dets allows the developer to store a collection of Erlang tuples in a file. Each tuple contains a key—by default the first element, but the element used as the key can be configured—and dets provides functions to retrieve tuples by key, delete them by key, and so on. dets tables may be either *sets*, in which case each key value may appear once in the table, or *bags*, in which case any number of tuples may contain the same key. Whether a dets table is a set or a bag is specified by an option when the table is opened.

Internally, dets organizes data as a linear hash list, which grows gracefully as more data is inserted into the table—hash buckets which become too full are split on demand. Space in the file is managed using a buddy system. The implementation is fairly complex, running to over 6,000 lines of code.

dets provides a rich API with over 40 functions, supporting table traversals of various kinds, including incremental traversals. Concurrent access and modification is supported too (although care is needed during incremental traversals). However, we restricted our tests to a small subset of critical operations, focussing on the insertion, retrieval, and deletion of tuples:

- `open_file(Name, Args) -> {ok, Name} | {error, Reason}`, which opens a dets file with the given name and options. The only option we specified was the table type, either set or bag.

- `close(Name) -> ok | {error, Reason}`, which closes the file again.

- `insert(Name, Objects) -> ok | {error, Reason}`, where Objects can be either a tuple or a list of tuples, which inserts the objects into the table named Name.

- `new(Name, Objects) -> Bool`, inserts the objects into the table named Name and returns true provided none of the Objects are already in the table.

... Reason}

MASTER THESIS PROJECT PROPOSAL

Contract support in automatic testing of
Java code

Your name here

Relevant completed courses:

TDA293, Software Engineering using Formal Methods

TDA545, Object-oriented programming

TDA550, Object-oriented programming continuation course



How we refer to a paper

J. Hughes, Why functional programming matters, *The Computer Journal* 32 (2) (1989) 98–107.

K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of ICFP'2000*, 2000.

How we refer to a paper

Authors

J. Hughes, Why functional programming matters, The Computer Journal 32 (2) (1989) 98–107.

K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In Proceedings of ICFP'2000, 2000.

How we refer to a paper

J. Hughes, Why functional programming matters, The Computer Journal 32 (2) (**1989**) 98–107.

K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In Proceedings of ICFP'2000, **2000**.

Authors

Year of publication

How we refer to a paper

J. Hughes, Why functional programming matters, *The Computer Journal* 32 (2) (1989) 98–107.

"Hughes (1989)"

K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of ICFP'2000*, 2000.

"Claessen and Hughes (2000)"

How we refer to a paper

J. Hughes, **Why functional programming matters**, The Computer Journal 32 (2) (1989) 98–107.

K. Claessen and J. Hughes. **QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs**. In Proceedings of ICFP'2000, 2000.

Paper title

How we refer to a paper

J. Hughes, **Why functional programming matters**, **The Computer Journal** 32 (2) (1989) 98–107.

Paper title

Publication venue

K. Claessen and J. Hughes. **QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs**. In **Proceedings of ICFP'2000**, 2000.

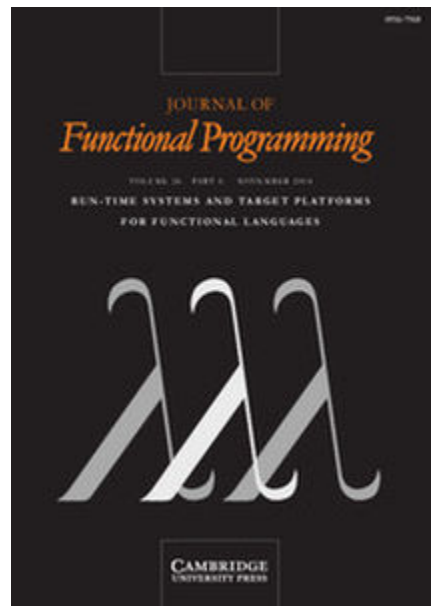


International Conference on Functional Programming

How we refer to a paper

J. Hughes, Why functional programming matters, The Computer **Journal** 32 (2) (1989) 98–107.

K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In **Proceedings** of ICFP'2000, 2000.





Types of publication venue

	Workshops	Conferences	Journals



Types of publication venue

	Workshops	Conferences	Journals
Page limit	6—10 pages	12—30 pages	unlimited



Types of publication venue

	Workshops	Conferences	Journals
Page limit	6—10 pages	12—30 pages	unlimited
Acceptance rate	$\geq 50\%$	10—30%	$\geq 50\%$

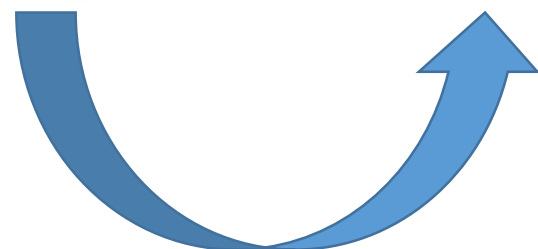
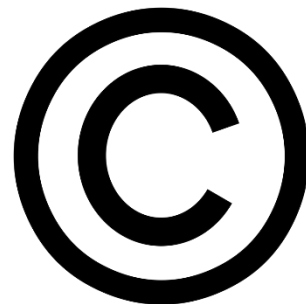


Types of publication venue

	Workshops	Conferences	Journals
Page limit	6—10 pages	12—30 pages	unlimited
Acceptance rate	$\geq 50\%$	10—30%	$\geq 50\%$
Time from submission to publication	3 months	6 months	years

Types of publication venue

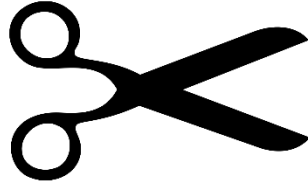
	Workshops	Conferences	Journals
Page limit	6—10 pages	12—30 pages	unlimited
Acceptance rate	$\geq 50\%$	10—30%	$\geq 50\%$
Time from submission to publication	3 months	6 months	years



30%
new



Conference page limit



What, Your Abstract, Mechanisms that Appropriately Prioritize...
 We begin by introducing the core of our abstract machine. In [2] we...
 2. Back ETC



Testing Noninterference, Quickly

Chitra Bhatia¹, John Hughes¹, Benjamin C. Pierce², Anind Sengupta-Zubovsk³, Dmitriy Vyukhin⁴,
 Artur Avramio & Amalendu¹,
 Lucinda Lopes¹,
¹University of Pennsylvania, ²Chalmers University, ³Microsoft Research

Abstract
 In some situations, it can be very costly to check the validity of...
 We begin by introducing the core of our abstract machine. In [2] we...
 2. Back ETC

1. Introduction
 Recent advances in the theory of...
 We begin by introducing the core of our abstract machine. In [2] we...
 2. Back ETC

} 30% new!





- The authors have carefully selected the most important material
- Best to understand the main points



- The authors included every possible detail
- Best if you plan to build on this work and need complete knowledge



“Direct to Journal”

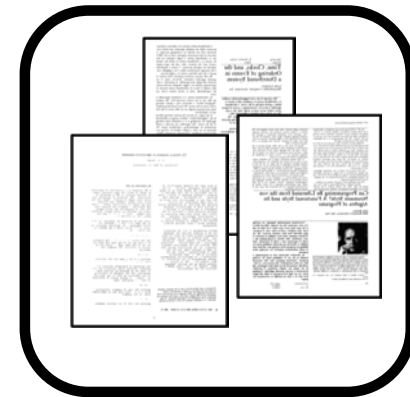
- Sometimes people just do that
- Paper needs to be longer than conference format
- Paper contains nothing new



Explain existing
results in a new
way



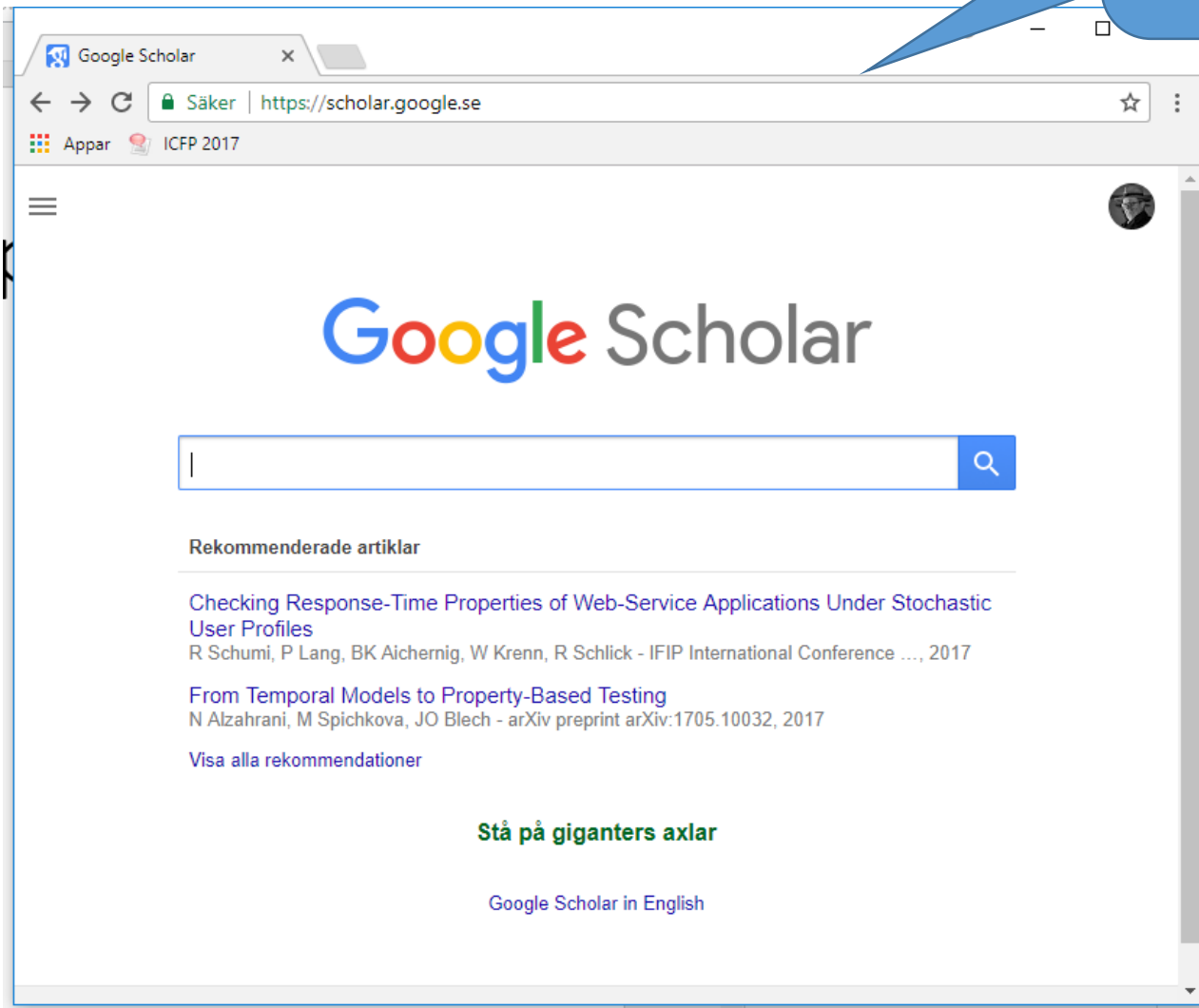
Surveys an
entire area



Integrates several
conference papers
into a coherent whole

How do we find a paper?

Google
"scholar"



The image shows a screenshot of a web browser displaying the Google Scholar homepage. The browser's address bar shows the URL <https://scholar.google.se>. The page features the Google Scholar logo, a search bar, and a section titled "Rekommenderade artiklar" (Recommended articles). Two articles are listed:

- [Checking Response-Time Properties of Web-Service Applications Under Stochastic User Profiles](#) by R Schumi, P Lang, BK Aichernig, W Krenn, R Schlick - IFIP International Conference ..., 2017
- [From Temporal Models to Property-Based Testing](#) by N Alzahrani, M Spichkova, JO Blech - arXiv preprint arXiv:1705.10032, 2017

Below the articles, there is a link to "Visa alla rekommendationer" (Show all recommendations). At the bottom of the page, there is a green link "Stå på giganter axlar" (Stand on giants' shoulders) and a link "Google Scholar in English".

How do we find a paper?

The image shows a screenshot of a Google Scholar search result. The search query is "john hughes why functional programming". The top result is "Why functional programming matters" by J. Hughes, published in "The computer journal" in 1989. The abstract discusses the importance of well-structured software. The search interface includes a search bar, a search button, and a list of results. Two blue callout boxes are present: one pointing to the search bar with the text "Search for authors and (part of) title", and another pointing to the right side of the search result with the text "Sources for the paper".

Search for authors and (part of) title

Sources for the paper

john hughes why functional programming

Scholar

Why **functional programming** matters

J. Hughes - The computer journal, 1989 - academic.oup.com

Abstract As software becomes more and more complex, it is more and more important to structure it well. Well-structured software is easy to write, easy to debug, and provides a collection of modules that can be re-used to reduce future programming costs. Conventional languages place conceptual limits on the way problems can be modularised. Functional languages push those limits back. In this paper we show that two features of functional ...

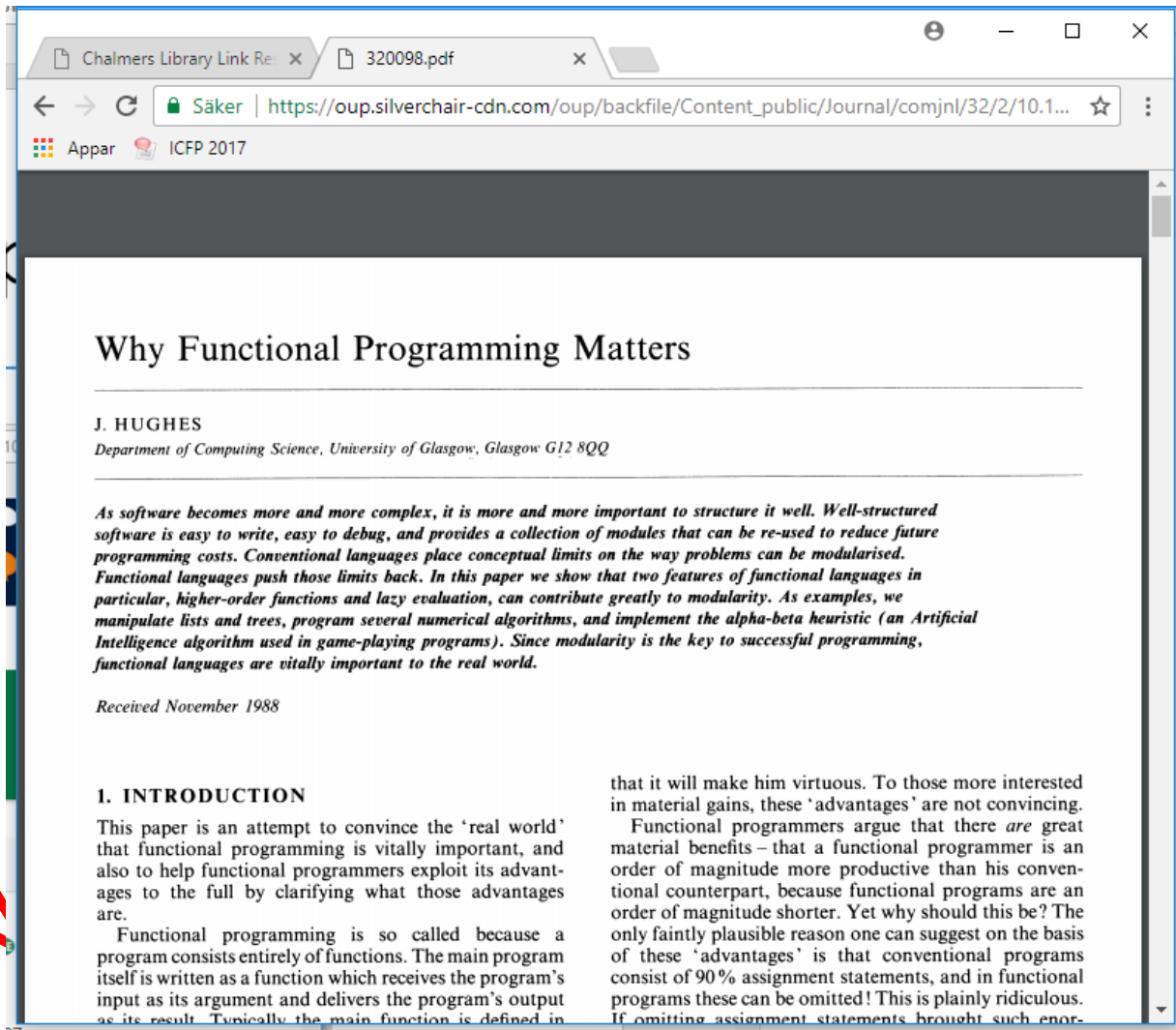
[PDF] oup.com
Full-Text@Chalmers

☆ 99 Citerat av 1112 Relaterade artiklar Alla 76 versionerna

Visar det bästa resultatet för denna sökning. Visa alla resultat

Hjälp Sekretess Villkor

How do we find a paper?



The screenshot shows a web browser window with the following details:

- Browser tabs: "Chalmers Library Link Re: X" and "320098.pdf".
- Address bar: "Säker | https://oup.silverchair-cdn.com/oup/backfile/Content_public/Journal/comjnl/32/2/10.1...".
- Page title: "Why Functional Programming Matters".
- Author: "J. HUGHES".
- Affiliation: "Department of Computing Science, University of Glasgow, Glasgow G12 8QQ".
- Abstract: *As software becomes more and more complex, it is more and more important to structure it well. Well-structured software is easy to write, easy to debug, and provides a collection of modules that can be re-used to reduce future programming costs. Conventional languages place conceptual limits on the way problems can be modularised. Functional languages push those limits back. In this paper we show that two features of functional languages in particular, higher-order functions and lazy evaluation, can contribute greatly to modularity. As examples, we manipulate lists and trees, program several numerical algorithms, and implement the alpha-beta heuristic (an Artificial Intelligence algorithm used in game-playing programs). Since modularity is the key to successful programming, functional languages are vitally important to the real world.*
- Received: "Received November 1988".
- Section: "1. INTRODUCTION".
- Text: "This paper is an attempt to convince the 'real world' that functional programming is vitally important, and also to help functional programmers exploit its advantages to the full by clarifying what those advantages are. Functional programming is so called because a program consists entirely of functions. The main program itself is written as a function which receives the program's input as its argument and delivers the program's output as its result. Typically the main function is defined in that it will make him virtuous. To those more interested in material gains, these 'advantages' are not convincing. Functional programmers argue that there are great material benefits – that a functional programmer is an order of magnitude more productive than his conventional counterpart, because functional programs are an order of magnitude shorter. Yet why should this be? The only faintly plausible reason one can suggest on the basis of these 'advantages' is that conventional programs consist of 90% assignment statements, and in functional programs these can be omitted! This is plainly ridiculous. If omitting assignment statements brought such enor-

PDF

How do we find a paper?

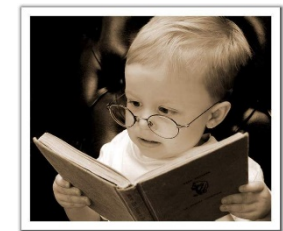
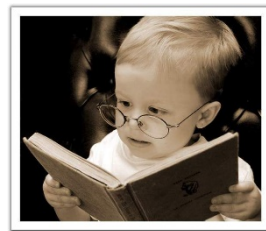
- Full-Text@Chalmers
 - Free access to *many* scientific publishers
 - Accessible from within the Chalmers network (including VPN)
- Digital libraries
 - ACM digital library, IEEE digital library, etc
 - (may require payment)
- Author's home page
 - Can often find a PDF by Googling
 - May not be the "official" published version

Open Access

Readers like it!

Authors like it!

- Publishers make articles free to download for anyone



The *reader* is the customer
Succeed by publishing a
quality product

The *author* is the customer
Succeed by publishing very
many papers

ABSTRACT

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

**Generate your own fake
paper at
[https://pdos.csail.mit.edu
/archive/scigen/](https://pdos.csail.mit.edu/archive/scigen/)**

Citations

...

results in higher-order functions.

Hughes makes a slightly different but equally compelling argument in **Hughes [1984]** where he emphasizes the importance of

...

REFERENCES

AASA, A., HOLMSTROM, S., AND NILSSON, C. 1987. An efficiency comparison of some

...

HUGHES, J. 1984. Why functional programming matters. Tech. Rep. 16. Programming Methodology Group, Chalmers University of Technology.

...

This insight has been described more appropriately by **Hughes**, Thompson, and surely others [**19**,33].

...

References

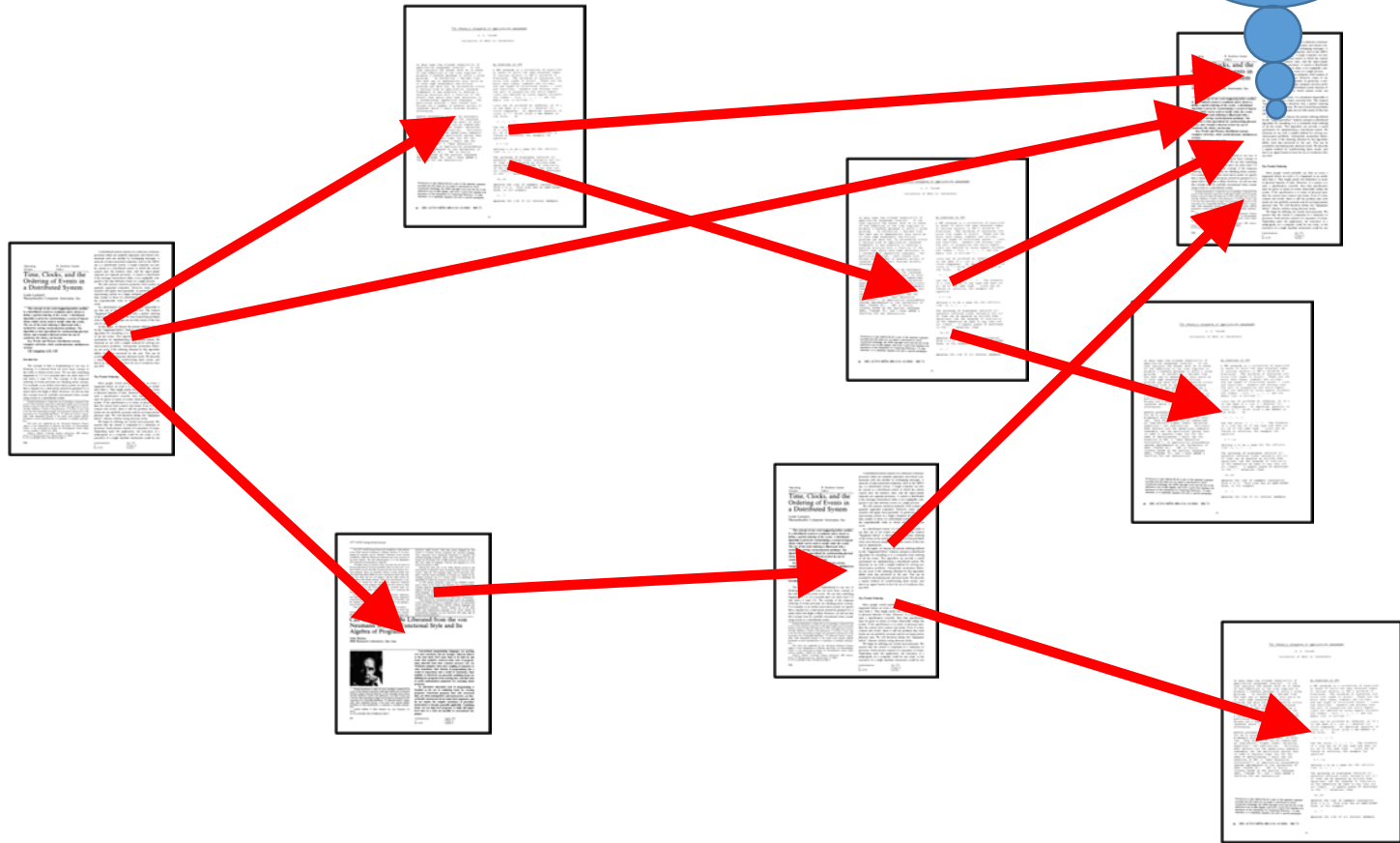
[1] L. Augusteijn, Sorting morphisms, in: S. Swierstra, P. Henriques, J. Oliveira (Eds.), ...

...

[**19**] J. **Hughes**, Why functional programming matters, The Computer Journal 32 (2) (1989) 98–107.

Citation graph

This looks like
an important
paper



Investigating a topic

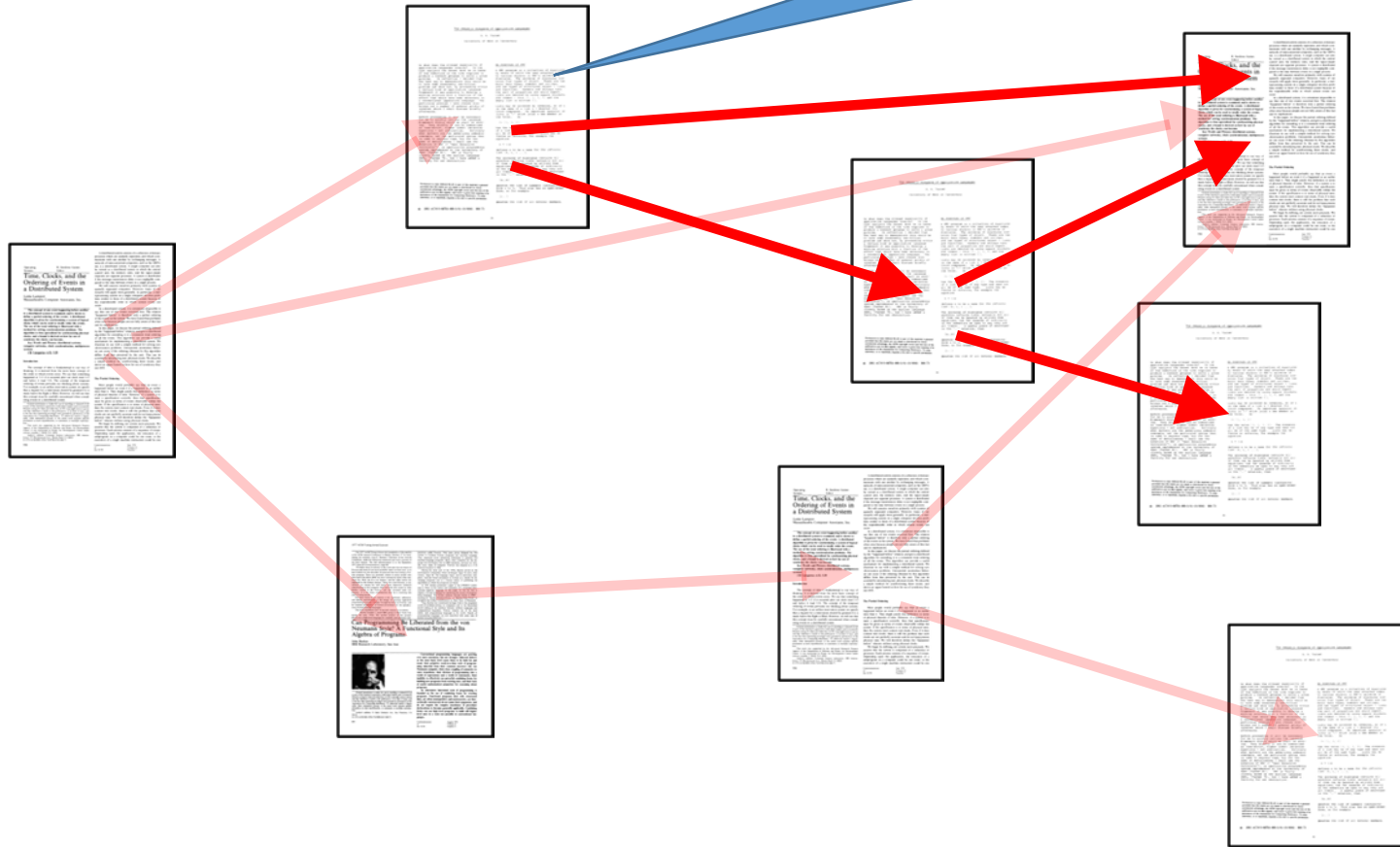
The papers that cite this one

Cited by 1112 other papers

The screenshot shows a Google Scholar search results page. The browser tab is titled 'Hughes: Why functional'. The address bar shows the URL 'https://scholar.google.se/scholar?cites=1662342056876...'. The search bar contains the text 'Scholar' and 'Ungefär 1 112 resultat (0,25 sek.)'. The main search result is for the paper 'Why functional programming matters' by H. Hughes, with 1112 citations. Below this, there are two other results: 'Structure and interpretation of computer programs' by H. Abelson, G. J. Sussman, and J. Sussman (1996), and 'Modern compiler implementation in C' by A. W. Appel (2004). The third result is 'ML for the Working Programmer' by L. C. Paulson (1996). The page also shows a search bar, navigation icons, and a sidebar with 'Appar' and 'ICFP 2017'.

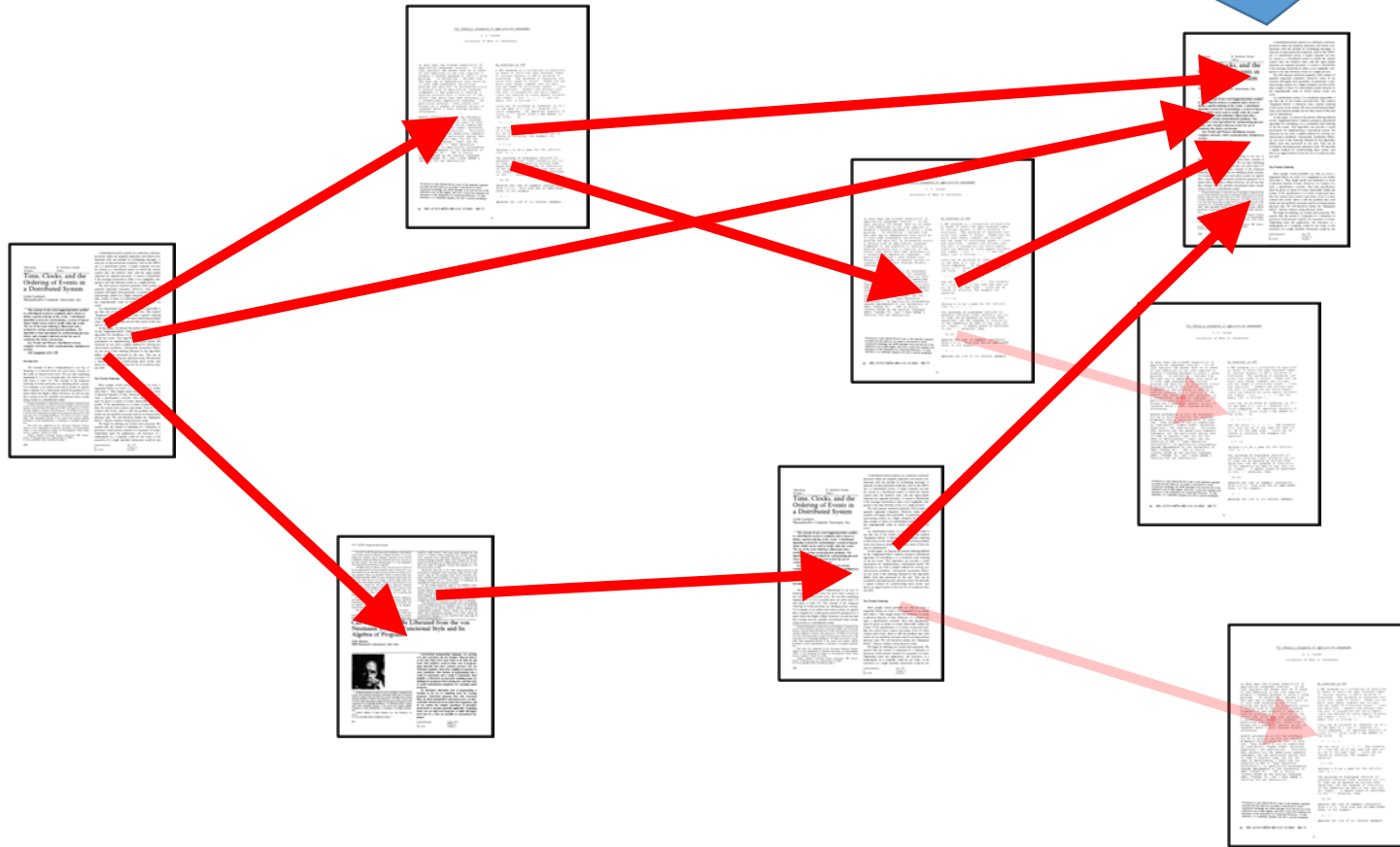
Citation graph

Follow references to understand the *history* of an idea

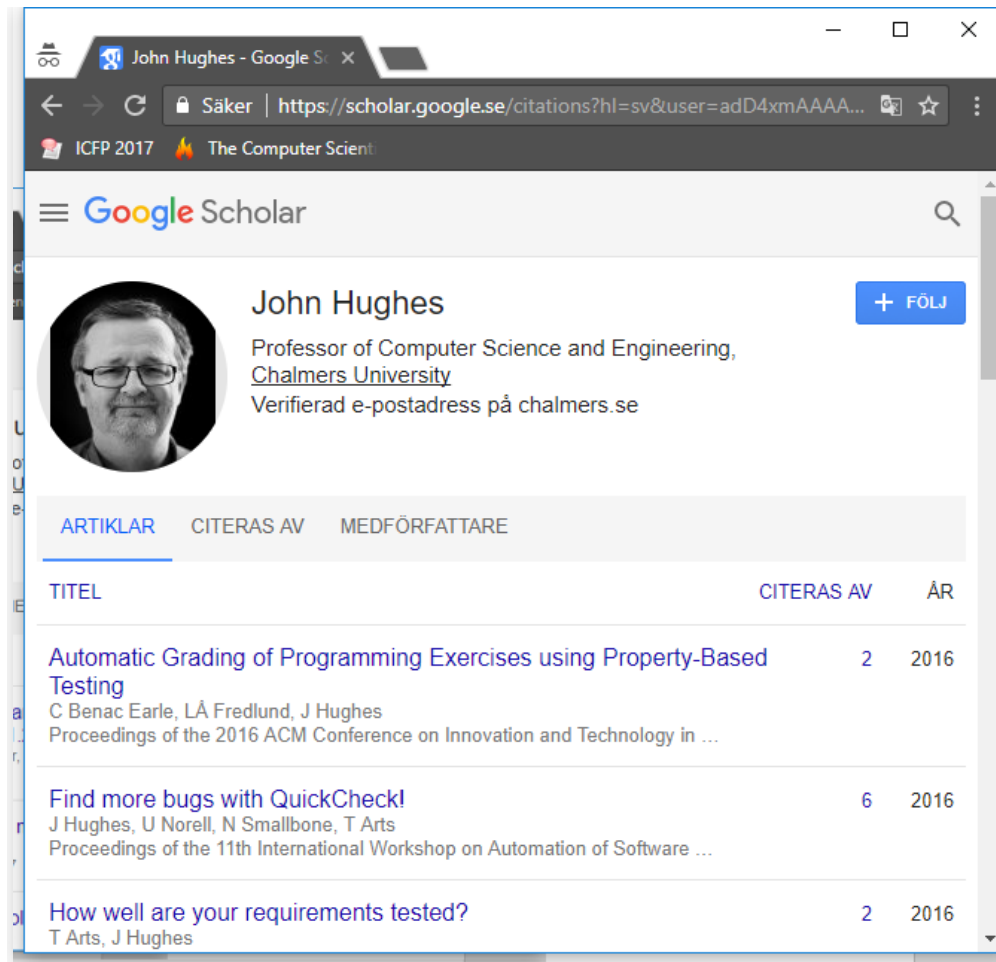


Citation graph

Use Google Scholar's reverse citations to understand the *impact* of an idea



What else did the author do?



The screenshot shows a web browser window displaying a Google Scholar profile for John Hughes. The browser's address bar shows the URL <https://scholar.google.se/citations?hl=sv&user=adD4xmAAAA...>. The profile includes a circular profile picture of John Hughes, a blue '+ FÖLJ' button, and his title as Professor of Computer Science and Engineering at Chalmers University. Below the profile, there are three tabs: 'ARTIKLAR', 'CITERAS AV', and 'MEDFÖRFATTARE'. The 'ARTIKLAR' tab is selected, showing a table of publications with columns for 'TITEL', 'CITERAS AV', and 'ÅR'.

TITEL	CITERAS AV	ÅR
Automatic Grading of Programming Exercises using Property-Based Testing C Benac Earle, LÅ Fredlund, J Hughes Proceedings of the 2016 ACM Conference on Innovation and Technology in ...	2	2016
Find more bugs with QuickCheck! J Hughes, U Norell, N Smallbone, T Arts Proceedings of the 11th International Workshop on Automation of Software ...	6	2016
How well are your requirements tested? T Arts, J Hughes	2	2016

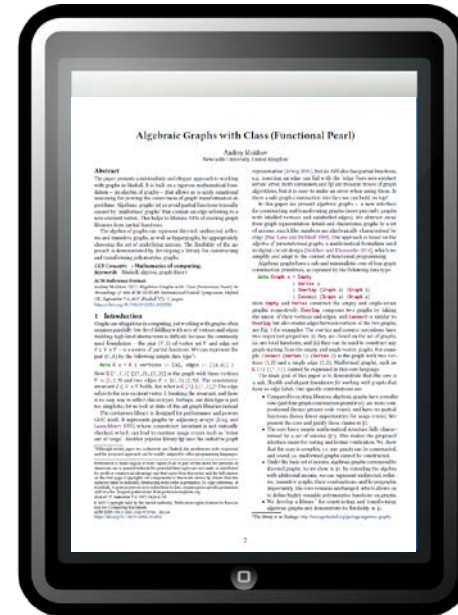
What makes a paper highly cited?

- An important, widely used result or method
- Inspired many others
- A good introduction to a topic
- A good survey of an area
- Easy to read and understand
- Old
- In a popular area

Why do authors cite?

- Present paper builds directly on a previous one
- Older paper solves a similar problem in a different way
- To show the author is aware of standard works in the field
 - Survey articles are invaluable!
- To cite author's own previous work
- To cite reviewers' own previous work

Scholarship



Often, the original papers on a topic give much better explanations, because that is their main *raison d'être*

How to read a paper

1. A quick scan

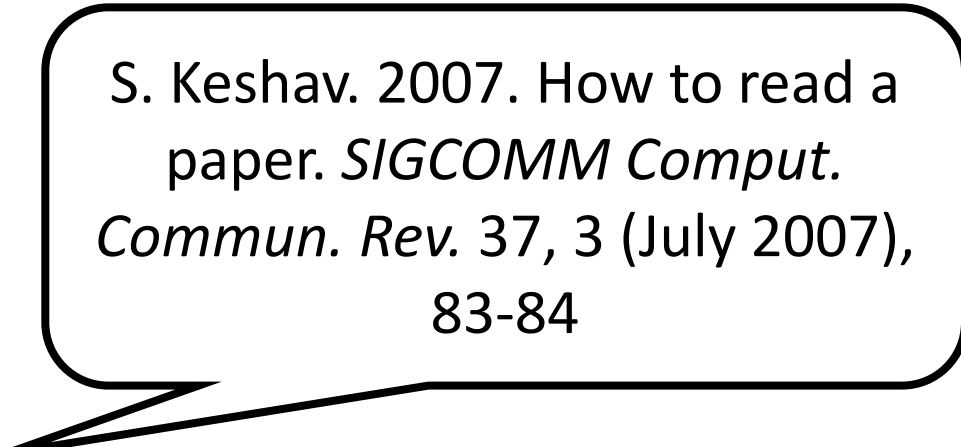
- What is it about?
- Do I want to read it?
- 5—10 minutes

2. Read

- Skip some details—proofs etc
- 1 hour

3. Reconstruct

- Make sure you understand every detail
- 4—5 hours



S. Keshav. 2007. How to read a paper. *SIGCOMM Comput. Commun. Rev.* 37, 3 (July 2007), 83-84

Your Task

Read Keshav (2007) first!

- Find an important CS paper to study
 - ≥ 100 citations
 - Read it thoroughly
 - Find and read *at least* one important previous paper (from list of references)
 - Find and read *at least* one important successor paper
 - Become an expert in the topic of your main paper
 - You'll be explaining it to the rest of us!
- 