My reviews of contributions to my Festschrift.

+ replies:

Roscoe	Abramski
	Welsh
	Milner
Tennent	Jackson
Lampson	Dijkstra
Dahl	Misra
Bjorner	Hehner
Stoy	Morgan
May	Brookes
Gguen	Zhou
Gordon	Bird
Gries	He
Knuth	Rabin
	Burstall
	Jones

all here

25 Jan 1994

Dear Bill,

Thank you again for your written contribution to my birthday book. It combines an excellent summary of the design philosophy which we shared in the research on CSP with a clear description of FDR, one of the most promising approaches to its practical application. And the exposition is exactly judged to tell me what I want to know about it. So my only comments are questions

Would it be reasonable to reference Hennessy and de Nicola's work on normal forms of CSP (in addition to [8])?

Would it be possible to apply similar techniques to CCS, and help to speed up the concurrency workbench?

(3) Would it help to compile low level processes to normal form – particularly if there are a lot of identical ones?

(4) Which are the eight coloured pegs in the game of solitaire (P-H. did not run to colour printing)?

(5) Solitaire is dual: any solution can be run backwards, interchanging pegs and holes. So the reservation of 8 pegs to the last ten moves is equivalent to requiring 8 specific holes after the first ten moves — an obviously drastic simplification.

(6) Solitaire is a monotonic game, in which earlier states never recur. Would it be worth while to allow an option to throw away earlier states, rather than merging them?

(7) These games would be nice examples for competition with BDDs, and other checkers.

(8) There is work on esctension of CTL model checking to time (including the duration calculus) in VERIMAG (Ahmed Bouajjani). They assume all time-specifications are linear, and use LCM.

(9) SDL seems to be the most popular specification language in the telecommunications industry. (much more than LOTOS).

(10) Could you generate specifications from a limited class of trace/refusal predicates? (MSc project?)

I couldn't quite assess (or even understand) all your proposals for improvement. From an academic point of view, the most interesting would be fuller explotation of compositionality: commercially too, it may give you the biggest edge over rivals.

Yours sincerely

Tony

25 Jan 1994

Dear Samson,

Thank you very much for your contribution to my birthday book. I am particularly pleased at your quotation from my traces paper. I also agree strongly with your own analysis of current problems in the field of semantic research, and your own approach to their solution. I have always used elegance of algebra as a criterion in the design of a single model. You (rightly) propose category theory as a criterion of elegance for an algebra, which you intend to be applicable to a whole range of models.

I too am pursuing a kind of universal philosophy, under the slogan "Programs are Predicates." The objects are the alphabets of

permitted free variables. A predicate can be made into an arrow in your symmetric monoidal category by partitioning the alphabet into two sets, the "input" variables and the "output" variables. The symmetric monoidal axioms say nothing more or less than "this can be done in any way you like." And one of the ways that you may like will give you standard sequential composition.

One of the advantages of starting with predicates is that each hom-set is endowed with a natural ordering (implication), which is equally useful in modelling non-determinism in implementations and refinement of specifications, designs, and programs. By endowing the predicates with the structure of a complete lattice, I get recursion as well. I am suspicious of more subtle definitions of non-determinism, which may involve more clever reasoning and less efficient implementation (eg backtracking)

When reading your work, I have often felt that if I studied it more closely I would be able to understand it. A study of this short article reveals that some of my problems are due to the author rather than the reader – terms and notations used without definition, unexplained deviations from the referenced "standard". I am greatly looking forward to studying the fuller account, and discussing it with you.

Yours sincerely,

Tony

25 Jan 1994

Dear Jim,

Thank you very much again for your contribution to my birthday book. It is a beautifully stated description of what must be the essence of any solution to the problems of engineering of software — or anything else of such complexity. You may be interested in one Company I have visited which made a policy of throwing away all design documentation, on the grounds that it is too often inconsistent with the code, so its use is actually misleading.

I am now consulting for a Company (a different one) which has to maintain a twenty million line program. Here the problem is that a typical upgrade requires change to

to a dozen or so disjoint modules. They hope to discover which modules are relevant by means of an execution trace! This problem seems insoluble by your methods, or by any other. All we can say is that the whole structure of the original program has been ill designed for the kind of future updates that are actually needed. We should reiterate Dijkstra's advice that a program should be designed as the "family of programs that it may evolve to."

Perhaps the basic problem arises because our program modules are in one-one correspondence with the intervals in main store which hold the code which is compiled from them. Object orientation may help to break up this correspondence. But maybe we need a new concept of "procedure call"

as an event in which more than two objects can participate. This is an idea taken from the definition of parallel composition of CSP; but I don't think anyone yet has tried to put it into practice.

Keep up the good work!

Yours sincerely

Tony.

25 Jan 1994

Dear Robin,

I very much appreciate the kind Foreword that you contributed to my birthday collection. It has always been my goal to win your respect and appreciation, especially since we often take opposite (though never opposing) approaches to the same problem.

In the past, I have always felt that kind words place upon me an obligation to live up to them. At last I can feel more relaxed, and enjoy them all the more—

Yours

Tony.

25 Jan 1994

Dear Michael,

Thank you very much for your excellent contribution to my birthday book. You are correct in your assessment of my appreciation of the broader canvas. And I agree strongly with the views that you so clearly and convincingly express. The link with reality is the most important thing; and only clear well-structured description can achieve that: notations, formalisms and calculation can start only after the link has been established.

I also strongly agree that a multiplicity of description methods must be brought to bear at different stages of the design, for different parts of the product, and for different classes of user. Like you, I deplore one aspect

of our current culture of software development, that managers, salesmen, politicians insist that all problems must be solved within a single framework; and too many researchers are willing (or coerced) to promise such a panacea. Where would mathematics be if all problems had to be posed and solved in graph theory? Or metallurgy, if molybdenum were the only metal in fashion? Until researchers are freed from the obligation to worship fashionable buzz-words, we will never be able to establish a rational structure for our discipline, with specialised notations and reasoning methods for different purposes, and with secure interfaces for combining them when necessary.

This is exactly where mathematical formalisation can make its most significant contribution; we can design embeddings which securely relate theories at different levels of abstraction; and we can clarify the restrictions and protocols by which different descriptions can be applied to different parts of a product. It is in the interfaces between technologies that errors are most frequent, most subtle, and most damaging. And that is not the only reason for Theorists to work on interfaces. Such work appeals to my curiosity about the underlying structure of our subject, and is essential if it is ever to be taught effectively at University level.

So I am really pleased that my views agree so well with those that you have more laboriously acquired from a lifetime of work in close contact with "the real world". I certainly

propose to devote what efforts remain to me to the "clear descriptions of complex systems in which many domains meet and interact". But more significantly, I want to encourage others to take up the good work. The encouragement that you give in essays like this will be even more effective.

Yours sincerely

Tony

25 Jan 1994.

Dear Edsger,

Many thanks again for your very nice contribution to my birthday book. I am taking a leaf out of your book (metaphorically, of course) in reading the articles and writing to the authors.

What a lot we have learnt from the relational calculus! I have just been rereading your last Marktoberdorf lectures as well. I have been hoping, with the help of Burghardt von Karger, to find a family grouping that would include more than three calculi.* Your design of calculational proofs is full of good ideas and good advice. But it didn't prevent me spending

---

*including ones without termcancellation or the cone rule

all day today on a wild goose chase!

My publisher organised a very nice party for me on the 12th, with a fair number of contributors as well as members of the Department. There are now 85 books in the series, so there is a good hope that I'll get to 100 books, before retiring.

Then there's Marktoberdorf. I hope to get down to planning my lectures straight away. I hope we remember this time to elect Richard Bird as Director. See you there.

We very much enjoy the quilt, which hangs in pride of place. Thank you, Ria, and also Ann, and best wishes for her 80th. We're planning for my mother's 85th.

Yours

Ian

25 Jan 1994

Dear Jay,

Thank you again for your beautiful contribution to my birthday book, and the kind dedication. I find your message totally convincing: this must be the right way to reason about highly parallel networks; and your exposition must be the right way of explaining it. The absence of pictures is most illuminating.

And the achievement is amazing: I have myself tried to explain "Fast Fourier Transform Free from tears" Computer Journal 20(1) 78-83 (Feb 77). Kruseman Aretz had done it better. I have read Dijkstra on Batcher's Baffler, and still been baffled. You have solved both puzzles, merely as illustrations of the attractiveness of your reasoning methods.

If you are still working on a non-preliminary version, may I suggest a little more work to make these two examples self-contained? The material on ~~~ Higher Dimensional Arrays seems to be preparation for something that never arrives: perhaps it should be saved for another occasion.

You have already had joy from this piece of work – I wish you more

Yours sincerely,

Tony

25 Jan 1994

Dear Rick,

Thank you very much for your contribution to my birthday celebration. It contains yet another of your brilliant new ideas, that it is amazing that nobody has thought of before. Of course, we should never have hidden time so thoroughly in Computing Science! Our programming languages really need a facility to specify $t := t + k\varepsilon$, where t stands for time, and k is the amount of real time that the programmer is willing to allocate before the next execution of a similar increment. The compiler (preferably) should check that the target can be met by the generated machine code, and reject the program if not. Because that is what the programmer actually <u>wants</u>. It is <u>much</u>

better not to start a program that it is going to take too long to finish!

The explicit mention of time makes an immediate entry to considerations of complexity. If you want to abstract from that, it is always possible to precede each program by an arbitrary selection of machine speed ε.

$$\varepsilon :\in \{\varepsilon \mid \varepsilon > 0\};\ \text{program}$$

I think this restores the theorist's "full abstraction" by reducing termination to a single bit.

There is another brilliant idea in your paper, that is the achievement of recursion without using a formal identifier for the recursive call! Indeed, it avoids semantic complication! It's also a very practical way of getting programmers to write assertions just where they are most needed! It really

achieves Floyd's original goal of Assigning Meanings to Programs.* Could you also solve the problem of first order parameter passing simply by allowing substitutions in the assertions which stand for the recursive call? If the compiler can recognise the substitution by pattern matching, generation of parameter passing by copy-in, copy-out could be automatic. But I don't think this will work for higher order procedures, where positional addressing of (a few) parameters is convenient.

I feel you have removed one of my mental blocks. I was always afraid of recursive equations of the form

$$X \Leftarrow F.X$$

because they lead to strangest fixed points, miracles, and partial correctness, of which I have long been ashamed. But your explicit

*new slogan: Programmes are Theorems!

introduction of t immediately leads to unique fixed points and total correctness, removing grounds for both fear and shame.

So I have much to thank you for: I hope I get as much reward from the other articles in the book, and that we meet again soon.

Yours sincerely

Tony

25 Jan 1994

Dear Carroll,

Many thanks for your excellent contribution to my birthday book. I have recently been enquiring after a definitive treatment of the relationship between relations and conjunctive predicate transformers; and now here it is.

And of course it is so nicely written, both in the details of the proof, and the overall structure of the development. It is like a detective story, with plenty of clues thrown in, challenging the reader to anticipate the writer. But the real pleasure and benefit is not just in the solution to the mystery but in the beautiful treatment of so many important topics along the way.

Thank you too for recalling the story from my first visit to Australia. I really do like questions from students, who are not in anything similar to flies, which I experienced on my second visit in good measure.

Yours sincerely

Tony

3 Feb 1994

Dear Steve,

Thank you very much for your evocative contribution to my birthday book. It reminds me strongly of the days when we were together exploring the foundations of CSP. One of our major achievements of that time was to factor out the question of fairness, and show that it was possible to write and prove useful parallel programs without relying on fairness of an implementation. I think this has contributed to the practical success of CSP, though clearly it does not appeal to theoreticians who devote their lives to fairness!

But your new work goes to show how we may get the best of both worlds:

Fairness if you want, and not if you don't.

Another deliberate feature of the design of CSP was the completely deterministic definition of the parallel combinator, showing how parallelism and non-determinism can be completely dissociated. It is only the external choice that introduces non-determinism. Did you have a reason for reverting to the highly non-deterministic || definition of CCS? Is it needed in order to get fairness? Or is it just an arbitrary decision – fashion perhaps? And was there a similar reason for omitting the pure Π-non-determinism of CSP?

I very much like the way in which you derive the "denotational" semantics of $\mathcal{T}$ as a collection of theorems proved from the operational semantics. That is surely the right (modular) way of establishing consistency

and/or completeness of two different presentations of a programming theory. Of course, with my ambition to supercede operational reasoning, and to start with specifications of observable behaviour, I would prefer to go in the other direction – to derive an operational semantics from the denotational; and I would like to share with you some ideas on how it might be done.

Then there is the third strand, the algebraic laws, which you have shown to lead to an elegant and useful normal form theorem for CSP. How are they affected by the introduction of fairness? Presumably you lose the wonderful strictness law for parallelism which you included in your improved model with Bill. But do you get something like Koomen's fair abstraction rule? It would be very interesting to compare the

algebraic properties of a fair CSP with the standard. Unfortunately, your language is a long way from CSP, so a direct comparison might be rather difficult. But I certainly wouldn't wish to encourage you to divert your research interests in this direction. There are already many excellent theoretical researchers who have devoted their efforts to fairness. Perhaps one of them may be inspired to pick up the challenge of your beautifully written article.

Yours sincerely

Tony

Feb 3 1994.

Dear Zhou,

Thank you very much for your inspiring contribution to my birthday book. It makes a substantial impact on problems of both theoretical and practical interest. Your introduction of mean values and events into the Duration Calculus is essential if we are to bridge the gap between continuous state-based specifications and discrete event-based implementations. Your examples give very convincing evidence of this.

The combinational circuit example suggests that the duration calculus might be useful in the detection and prevention of spikes, which can be a bit of a problem in some circumstances. Ian Page had a student, Brian Thompson, who did some commercially valuable work in this area, using four-valued logic, just the same as your

$\uparrow P$, $\downarrow P$, $\bot P$ and $\top P$.

In the familiar example of timed automata, I would like to see if we could use the duration calculus actually to write abstract programs. Ben Moszkowski presented a beautiful example of Interval Temporal Logic used in this way. at our recent meeting in Copenhagen. For example, I would like to express our very first design theorem for the gas burner as.

Spec: $\neg \Diamond (\ell \geq 60 \wedge \overline{Leak} > \frac{1}{20})$

Des: $\left((\ell \geq 60 \wedge \overline{Leak} = 0); \ell \leq 1 \right)^{\omega}$

Theorem: Des $\Rightarrow$ Spec.

Des is expressed as an infinite loop, which should be conjoined to the rest of the program, and then merged with it, using the interleaving laws of the Duration Calculus. This will produce a program that is correct by construction.

I have always regretted that the next design decision in your reference [9] was expressed as a flow chart, because of my bad experience of their influence in computing science. I think you pay the price in pages of formal material, expressing the linkage between your design and the otherwise meaningless phase names that appear in the flow chart. Would it not be possible to take a much more "structured programming" approach, and develop the program gradually from its specification as a simple infinite loop? Or perhaps devine it by merging a conjunction of infinite loops, each derived from little aspects of the specification, in the same way as the theorem shown above. I expect that is what you mean by "refining real-time requirements"

Burghard von Karger is working on the basis of a calculus which will codify the common properties of the relational calculus with

those of models of free groups, free monoids, regular expressions, intervals, traces, and even CSP. Your axioms will be very useful: I wonder how many of them have wider validity, and how the others will have to be qualified in the more general calculus.

Also in Copenhagen we heard of some interesting model-checking work at VERIMAG on extending a CTL checker to deal with durations. I wish you could have been there: I hope we can arrange for a visit later this year.

I have left the best bit to the last: many congratulations on your election to the Academia Sinica. That must be a great boost to Computing Science in China! And I hope that your stay in Macau is getting more congenial to you and your family.

Yours sincerely,

Tony.

Feb 3, 1994

Dear Richard,

Thank you very much for your highly elegant contribution to my birthday book. I think it is a clear demonstration of the maturity of your calculus, which goes far beyond anything that I could have expected when I made the naive suggestions that you so kindly acknowledge. You now have a well-tempered clavier, and can look forward with pleasure to the forty eight preludes and fugues.

This article is a masterpiece in its own right. Seven pages to set up the calculus, and the same number to derive Earleys algorithm. Not a word is out of place. But its not easy for me to follow. The explicit mention of types in category theory gives a peg for the understanding: you eschew that. The use of diagrams to illustrate uniqueness reasoning often gives a clue to the shape of an

it too. So the only recourse
itions. But this should be
ie. There must be a way
concepts which does not require
For example, a catamorphism
whose structure is the same
it is applied to. – including the
, inverse catamorphism is structured
ult. Your proposition 1 is
t of the coroutine principle,
need to produce the whole
then consume it all – you can
– a time, using a single pattern
If this sort of insight is correct,
some significant names for your
rems – ones that will appeal outside
mmunity?

Another place where such intuitions are important is when you make a negative claim "Unfortunately no such identity holds." Quite a shock when everything has gone so smoothly so far. How should I check such a claim? An answer to such a question might convey useful insights.

You are justifiably pleased by the smoothness of the transition to executable notations when the time is ripe. It would be very interesting to explore the transition (in suitable cases) to other executable paradigms, for example systolic arrays or logic programs. You have made such a successful separation of concerns: there is good hope that your methods of reasoning about specification at a high level of abstraction will bridge the culture gap between various schools and paradigms of programming. That will be a great deal more practically useful (and infinitely more intellectually attractive) than

current efforts to achieve a synthesis at an operational level. That way lies the Turing tarpit!

Please pass on my thanks in full measure to Oege.

Yours sincerely

Tony

Feb 3 1994

Dear Jifeng,

Thank you very much for your important contribution to my birthday book. It is a convincing model of hybrid system description, combining the merits of the duration calculus approach with CSP. Your treatment of differential equations seems to enjoy Ben Moszkowski's "very compositional" properties, suitably adapted for continuous time.

The two examples at the end are an excellent example of the strength of your approach. The algebraic transformations follow closely the engineer's operational understanding of what is going on. This kind of symbolic execution should be eventually subject to mechanical elaboration.

Some of the laws that you use in the examples seem to be rather specific to the examples themselves. Eventually, people who enjoy algebra may like to show how they can be elegantly derived from the more basic laws. Your treatment gives them excellent motivation for doing so.

Over the last ten years I have — as you know — very much enjoyed working with you on this kind of investigation; and it is a great pleasure for me to see how far you have progressed both in the formality of the work, and in the clear and well-motivated style of its presentation. This article is yet another confirmation of your leadership in the research; and I welcome the prospect of following and contributing to its advance.

Yours sincerely,

Tony

2 Feb 1994

Dear Michael,

Thank you most warmly for your very nice contribution to my birthday book. It is a lovely example of the power of probability in a distributed computing system. It is very clearly presented, with all the assumptions made clear and fully justified. It is amazing how distribution permits formulation of computing problems that cannot be solved by ~~com~~ Turing machines, though philosophically I do not know what to make of it.

David Deutsch has a very simple example of the power of distributed quantum computing to compute the square root of Boolean negation. That is a transducer f such that $f(f(x)) = \neg x$. But perhaps that's a bit of a cheat, since he transmits a superposition of signals between

the two fs. I could do the same digitally with a two-bit representation of Booleans.

My enthusiasm for probability in distributed programming remains as keen as ever. The ease of independent sampling without any coordination is the key (imagine having to share a single random number stream — all the joy would go out of it). I'm very impressed ~~ab~~ by the simple self-stabilising algorithms devised by Ted Herman and Geoff Brown and J. Rad.

I wonder whether you are right in ascribing probability to truly distributed implementations of CSP. It was certainly my intention that the non-deterministic resolution of distributed waiting should be made by a demon (your adversary) — even one who is not obliged to be fair in the scheduling of processes. Even so, the

programmer can design terminating computations, based wholly on worst case assumptions, without any appeal to fairness or probability. I thought this separation of concerns was a valuable property of CSP in practice; though of course it is useless for the conduct of research into fairness! Your remark about "termination with probability one" is ~~it~~ justificable, if only by the hardware property of the arbiters needed in a distributed of the external choice □, that one can only put probabilistic bounds on the amount of time that they may take.

I enjoyed reading your proof. It is an excellent reminder of the pattern of real proofs in mathematics, and even more in science: paint a word-picture of something, and then tell a convincing story about it. That is the way all real startling discoveries are made and reported. After you come the formalisers, trying to turn

your brilliant intuitive discovery into a reliable routine for ordinary programmers. That is what my colleagues at Oxford are doing with probabilistic CSP. But that is much too accurate. It is suitable for defining discrete event simulations, (which are unfortunately quite unsuitable for distributed implementation). Again, it is J. Rao and Jay Misra who have a calculus suited to "probability one" type of reasoning; like you, they clearly differentiate this from the worst case adversarial reasoning of demonic non-determinism.

Thank you for this discussion, and for your card. The heart project did not quite die here. Denis Noble was quite pleased with what Richard Burrows did for him — showing the effectiveness of parallel computation of a single linear strand of nerve cells. But it's still a long way from the experiment you would design.

Yours sincerely,

T.

2 Feb 1994

Dear Bob,

Thank you for your flattering and admirably clear contribution to my birthday book. I greatly enjoyed reading it for the second reason alone (well, to be honest, for the first reason as well). Although I have worked intensively on data representation (at two widely separated periods), I am woefully ignorant of the literature; your copious references are an eye-opener.

I had not realised that local variables were such a problem to abstraction: though I am not surprised, if they are given such an operational semantics that you describe. In predicate models of programs, locality is represented by existential quantification (which is also often interpreted in "possible worlds.") Do similar problems arise in higher order

predicate logic? If not, could we look for our solution there? I suppose we have to model the additional fact that local variables (declared since the procedure) must remain unchanged.

The problem of "lifting" a predicate to a new environment is one that occurs often in specifications; and you must be right in suggesting that a liftable test (program or predicate) should have a meaning that already includes all environments that it may be lifted to. But your insight is that at the same time you can restrict the environments that are "natural" in the sense of respecting data abstraction and invariance of locals. Very neat!

But isn't it a bit of a "cheat"? Couldn't the same technique be used to justify quite absurd methods of data abstraction (relationships between powersets of the two data spaces, for example). The construction is so powerful that it is difficult

to see what you get when you apply it. I expect you have given the answers in [26].

When I took a categorical approach to data refinement, I was particularly interested to find that everything still works for a non-deterministic programming language. I was able to deal with higher order procedures and arbitrary copying only by restricting the linking relation R to a retraction (projection-embedding pair). My colleagues have since shown me that in the category of monotonic predicate transformers, every relation is a retraction. Using this idea, Paul Gardiner was able to prove completeness of relations as a means of establishing refinement of an object (packet of procedures, &c. sharing a hidden local state). I wonder whether that kind of result generalises to the deeply nestable world of ALGOL?

My interest in data representations is now in a revived state, because I have been using them with my colleagues in the relatively

practical task of proving the correctness of compilers. I hope to reduce the proofs to mere algebraic calculations, using algebraic laws based on the semantics of the language. Augusto Sampaio found procedures a bit of a pain, because of the low level of abstraction which we used in the semantics. And we didn't even try higher order procedures! This might be a suitable application for your new results.

Thanks also for the point you make in your discussion. If I had such a thing as a mission in life, you could say it was to rescue the world from the tyranny of operational reasoning.* I think I can cite your more abstract definition of variable declaration as an example of that.

Yours sincerely,

Tony

*which explains my antipathy to operational semantics

2 Feb 1994

Dear Butler,

Thank you very much for the elegant piece that you contributed to my birthday book. I remember you were already working on this topic in 1988, when you lectured on it to the Marktoberdorf Summer School. It has certainly repaid your study. Not only is it of current practical importance in the implementation and use of shared-store multi-processors. You have developed it into an object lesson in the formal analysis and presentation of a complex algorithm.

Your first achievement is the organisation of the structure of a family of solutions, with a clear explanation of exactly where and why they diverge. Your use of boxing and crossing out

is very effective. Your state machine model and notation are clear and simple. You seem gain by dissociating the hidden actions from the external actions which they enable. This is design style which is new to me. – indeed I have suffered from a prejudice against fairness that seems to be needed to make it work. But you are right: the important design decisions and correctness arguments can and should be independent of fairness

Your definition of simulation is very close to half of Milner's definition of weak bisimulation, and is clearly appropriate to the purpose. I would have been worried about infinite sequences of internal actions, but wrongly so. How wise you are to put liveness beyond the scope of this paper. I suspect you have good reasons for

believing that in this particular application area, liveness is not an issue. For example, in hardware there is not much penalty in performing a hidden action on every (non-active) cycle of a synchronous circuit.

Your ideas for smooth combination of algorithms written partly in software on top of cheap and efficient hardware are most appealing, and deserve to be put into practice to solve immediate problems. One of the reasons why parallel processing is such an attractive and fruitful area of research is that there is a real prospect that programmers who adhere to certain precepts of abstraction and structure will be rewarded by programs that are not only clearer and more reliable, but also run more efficiently on a variety of parallel architectures. We have often discussed how

commercial pressures and brute ignorance of hardware and software designers seem to conspire against realisation of this dream. The trouble with our industry is that anything that they don't get right first time remains wrong forever, because of the deadly effect of legacy.

Well, that gives us even more time to perfect our ideas by research, and propound them in teaching. In these respects, your article will be seen to be a landmark, one of those rare achievements in formalism where the ratio of effort demanded from the reader and contributed by the writer is close to zero. — as it should be, if our science is to advance.

Yours sincerely

Tony

31 Jan 1994.

Dear Ole-Johan,

Many thanks again for revisiting Monitors. You have rightly seen how the wait and signal primitives were modelled on your ideas of semi-coroutines. But I always knew there was some unfinished business here, and you have finished it off admirably, with a succession of four entirely convincing strategies. And now you have reached an excellent place to stop.

You have also suggested the path of further development of my own thought. As you point out, proper understanding of properties of scheduling requires an account to be maintained of the history of the interactions of the object with its caller. And that way lies CSP.

What I found with CSP is that a great many exclusion and scheduling problems could be solved (or even: never arose) without constructions as elaborate as Monitors. My original inspiration was found in SIMULA 67 and in working on PASCAL PLUS, and I always intended to develop CSP as a model of concurrent object-oriented programming. When I got round to it in Chapter 6 of my book, the actual constructions seemed more laborious than I wanted. I was unable to reproduce cleanly the nice property of the monitor, that it never needs to know how many clients are using it. Some more unfinished business here, perhaps.

One thing that discouraged me about monitors is that they seemed to be so small! They have a rather insignificant part to play

in the large scale structuring of a parallel system. Their chief merit was their contribution to hierarchical structuring of "virtual resources", and their extremely efficient implementation in a shared-store machine. At the time, these were not so fashionable, but now perhaps the idea should be revisited. Thank you especially for doing so.

I was delighted to see you again in Copenhagen, and hear you in those excellent trios. I look forward to your comments on the PROCOS work, if any of it takes your fancy, or just excites your curiosity

Yours sincerely,

Tony

31 Jan 1994.

Dear Dines,

Many thanks for the greetings from Macau, from yourself, your colleagues, and your students. I got them by e-mail, by fax, by mail, and eventually by personal courier, who also brought the symbolic dish. I shall treasure it with the other mementoes of the kindness of my friends.

Thank you too for your inspiring contribution to my birthday book. It is a welcome reminder that Computing Science is indeed a Science, and that our task includes the description of reality, as a prelude to understanding it and then controlling it. Michael Jackson's Introduction says what I mean much more persuasively than I can. You agree that the distinctive contribution of Computing

Science is that we have to describe complex systems as a whole rather than just concentrating on parts. We have to elucidate structuring principles, within which detailed design of the parts can be safely undertaken, with interfaces so clearly defined that we are not overwhelmed by unpleasant surprises during integration testing.

And we should do this not just for a single product delivered on a particular day, but for whole families of products, implemented on different equipment, on different scales, and at different times during the life of a product range. Of course, the same problems face all branches of large-scale modern engineering. What is the specific contribution of Computing Science is to be extremely explicit in our approach to the task — you could even say "scientific."

We hope to use standard concepts of pure mathematics, with explicit appeal to abstraction, in a way that can be taught as part of an academic education, and form the cultural framework for professional discourse of engineers.

And real engineers work on real things like robots. If we are to subject our ideas to scientific test and refinement, these are the case studies that we must undertake and publish. So I am delighted that you have chosen this occasion to publish an account of your trail-blazing efforts in this area. I hope it will achieve all its short-term objectives, because they will surely contribute to the long-term objectives which we all share.

Yours sincerely,

Tony.

31 Jan 1994

Dear Joe,

Thank you for the very interesting contribution to my birthday book, and the kind remarks that you make. The Id language is an exciting development in programming language research, because of its explicit structuring into three layers (like the psychoanalyst's superego, ego, and id). That seems to be the only reasonable way of defining a wide-spectrum language, for use at many differing levels of abstraction.

Ideally, one would like to use the efficient but risky M-structures at the lowest level of an application, to build a secure platform on which large-scale structuring could use the cleaner and safer concepts of functional programming.

Unfortunately, the sordid reality of programming stands this pretty picture on its head: the functions are most useful for the components, but the large granularity of applications (particularly on current parallel architectures) forces the use of M-structures at the most global level. I doubt this problem can be solved by language design: though maybe the particular model of shared store selected by Id is not the easiest to use! or to implement efficiently.

I was delighted also to see how you used CSP to model the semantics. I always had a sneaking hope that CSP theory might be found useful as a "target language" for semantic definitions of parallel languages, in the same role as the traditional lambda calculus. Chapter 6 of my CSP book is my attempt

to do that for a parallel object-oriented language, where accounting for unused indices was indeed a bit cumbersome.

Your use of CSP in the semantics is very straightforward, and appeals to my "operational intuition" — according to Dijkstra, the lowest level of my soul. But then you go on to give an even more elegant conventional treatment of the strict/non-strict semantics. Where have all the problems gone? I suppose it's in the definition of the barriers. One test of a formal semantics is its use in deriving methods for development of correct programs, for example algebraic laws, predicate transformers, or verification conditions. For this, I think I would prefer to start with the strict/non-strict semantics.

Is this the fault of CSP? I think rather that it is the success of the functional

levels of Id, which have successfully achieved a degree of abstraction that can be neatly formalised in a functional style of semantics. Your proof of embedding crosses a significant gap between abstract and concrete, and shows that Id has also been successful in merging its two higher levels with its lowest.

Researchers in CCS are engaged in a similar project to use the new π-calculus a common foundation for functional and parallel programming. I think I'm allowed a little pleasure in congratulating you for getting there first.

Yours sincerely

Tony

31 Jan 1994.

Dear David, May.

Thank you for your wide-ranging contribution to my birthday 'book'. Now I can design not just one parallel computer, but a whole family of them! But I'd rather leave it to you: your arguments are completely convincing that you can lead us towards an exciting future.

In a few points I am perhaps more pessimistic than you. My experience with Oxford Parallel suggests that it will be not easy to increase problem size to exploit more processors. Existing problems have been adjusted to fit existing computers. The minds of those who program and use them have become limited to match current performances. As always in computing, the successful breakthroughs have to meet new needs, and create new desires. To

synchronise this with new technology is the goal of an explicit commercial strategy. The advance in communications may help — computers can send each other enormous numbers of "requests for quotation," and everyone will be compelled to keep up! Don't underestimate the importance of data bases, queries, updates and accesses needed to process these transactions.

I found it difficult to visualise some of your ideas about shared memory without knowing roughly the page/line size of your caches. Won't the compiler need to know that? If so, I fear re-emergence of the kind of hardware/software linkage that virtual memory is supposed to save us from. Don't you need to weigh the advantages of long cache lines for sequential access, and for deciding the unit of hashing for exclusive access?

It was your section on concurrent access that I was most concerned about. It seems

rather unlikely that manufacturers will agree a set of reducing operations simultaneously in their hardware and their programming languages. Even so, I would expect that the most immediate and severe problems will arise from concurrent write, because the majority of software designers to date have built all their scheduling and synchronising mechanisms by hammering a few locations of shared store. As usual, they will blame the hardware rather than their own ignorance and lack of foresight.

Nevertheless, virtual shared memory must be the way of the future. It's the only paradigm that software writers ~~who~~ won't mess up with layer upon layer of overhead. Instead, they will mess themselves up with non-deterministic interferences and deadlock.

To save myself from cynicism, I have to laugh at the irony of it. And it makes me value even more the clarity of your own thinking on the subject: it is our best hope of escape from the legacy of the past. Even on the most powerful shared memory computers, there is virtue in synchronisations which involve the least possible number of processors (namely two) and storage use which similarly involves the least number (in this case, one!). It is ironic that current BSP thinking has gone to the other extreme in both respects. Fortunately, it compensates by seeking the largest possible grain size — the outermost loop on each processor. If your hardware can deal efficiently with both extremes, there is good hope that will be good enough. I look forward to seeing the fruits of your design.

Yours sincerely —
Tom

28 Jan 1994.

Dear Joseph,

I want to thank you again for your marvellous contribution to my birthday book. It is very clearly written, at exactly the right level for me to understand and appreciate it. I particularly admire the combination of the highest level of generality with meticulous attention to detail. My temptation is always to say "and the results obviously extend to multiple types and..." and that is often a mistake. Even if the obvious extension turns out to be valid, it is only a result of meticulous care in the definitions and proofs — and sometimes there are side conditions which are far from obvious indeed, and which need to known by those attempting proofs.

My own inclination is to tackle very specific modelling situations, in which the linking homomorphism can be defined explicitly, and even then it's hard enough. As always, the best approach to simplify the task is to generalise it; and that will simplify other related tasks as well. Your algebraic approach immediately generalises to whole classes of models and homomorphisms between them; and more specific representation proofs also can be applied to whole members of the class. In many cases, you define a canonical method, the most general of its kind. And finally, your proof methods are primarily algebraic, which offers the best way ahead for both human understanding and mechanical assistance.

There is one further generalisation suggested by Messeguer, to replace equations by inequations throughout (in a consistent direction of course), and equivalences by preorders. I was astonished how easy this was when I applied it to data

[illegible] But perhaps

you have tried it, and found something that does not go through.

The first unexpected definition that I encountered was number 29. Partial implementation (if I interpret it correctly) allows two (or more, or all) abstract values to be represented by the same concrete value. This is exactly the opposite way round to my abstraction function, which maps many concrete values to a single abstract one. Yet the equivalence required by total implementation seems unnecessarily strong, particularly if the language provides no method for testing equality. Also, partial correctness requires most of the reasoning to be conducted in the concrete algebra, which one would expect to be more messy. But I have no reason to suppose that your definition will not turn out to be the right one after all — everything so far has been!

I wonder if I would gain confidence if your definition were more explicitly linked to its implications for models. Paul Gardiner

and Carroll Morgan have some quite powerful results about the completeness of data refinement methods in predicate transformer models of mutable object behaviour. It would be very interesting (to me) to enquire what the corresponding algebraic concept would be? Even if it's not the method of choice.

I found only one misprint – in the type definition of "top" on page 135. And maybe I had unnecessary difficulty in understanding ! on page 138. Is it not merely a hint to the implementation that pointer copying would be an invalid optimisation in a mutable environment? It is totally irrelevant to the mathematics, which would remain the same if <u>all</u> duplication were done by copying.

I have spent a very pleasant day in your company, without you knowing it. Pass on my thanks in due measure to Grant

Yours sincerely,

Tony.

28 Jan 1994

Dear Mike,

Now that I have read your contribution to my birthday book, I really must thank you again. I am amazed, delighted, and of course very flattered that my 1969 article is still inspiring such beautiful developments as yours, which completely solve problems of partiality, which I was too lazy to tackle at the time.

And what you have achieved with your mechanization is far more than I would have expected. You have not only proved the correctness of a compiler, but a verification condition generator as well. When my friends at CLinc confessed that they had no proof of consistency of their VCG with the operational semantics used to

prove compiler correctness, I thought this would a very difficult gap for them to fill. But that is just what you have done! Or is there still some gap that I haven't seen? In section 9.7, perhaps, which I didn't altogether follow?

My own approach to the same problem is of course not based on mechanization, but on an attempt at extreme modularity. For example, I try very hard to avoid constructive definitions like those at the top of page ~~19~~ 149, which are then incorporated in a proof by structural induction. I would rather give some independent meaning to the notations, clearly stating what it means for them to be correct. Then each of your definitions is proved independently as a theorem. I like to think this is more modular in the face of language extension — though clearly it would be disastrous if I had to change the definition of correctness. My real motive may

less creditable — that it provides me with elegant material for a keynote address!
It is quite clear that the idea and substance of the proof is unchanged when you replace definition by theorem and vice versa. And your choice is probably the best for mechanisation. It will be very interesting to see.

In 9.5 and 9.6 you give two different versions of the axiomatic semantics of the language. 9.6 would have sufficed, perhaps. The invariant I is at present not paying its way. When it does, you may have to weaken the 9.5.5. consequence rule to insist that it remains unchanged. Why do you not keep time as an expression containing ^(initial values of) program variables; you could use a conditional instead of Mosc in the conditional rules, and avoid the curious [T] [F] annotations. Alternatively, just make time into a normal program variable*

I met Ben Moszkowski again in Copenhagen last week. He has done some great things with temporal logic, and he

---

*see the lovely article by Rick Hehner in the book.

made an excellent presentation. I have been trying to persuade my colleagues in Copenhagen to write abstract programs in the Duration Calculus, following Ben's lead. But they seem hooked on the state-chart culture, where every arrow has to be dealt with separately. Is this also the basis of your doubt about the STA formalism?

Yours sincerely

Tony

28 Jan

Dear Helen and Jacki,

Thank you very much for the very enjoyable party. Although I knew nothing of it in advance, and would not have been able to tell what I wanted if asked, it turned out to be exactly the right thing, both for me and Jill, who joins in the thanks.

Thank you too for the lovely book of Japanese prints. They are splendid: I hope you had a good look at them before wrapping it up. I've seen three western small exhibitions recently, in California, Copenhagen and in the Ashmolean, — but not in Tokyo! The book trumps them all.

And so does the other book, the essays. I have been reading them with great pleasure and illumination, each of them a masterpiece of its kind. I have been writing to the authors to tell them what I have gained from the study, in the hopes of following up some questions too. I hope they all appreciate each others' contributions too. And of course that it finds a much wider audience, as it deserves.

Yours sincerely,

Tony

Dear David,

It is always a pleasure to read an article by you, and your choice of Quicksort is much appreciated in your contribution to my birthday book. Thank you very much.

I sympathise with your frustration, but regard it as inevitable. In fact, it's an excellent idea for any professional to explain things in rough sketches, analogies, hints, handwaving etc. But it's only a good idea because it can be backed up by detailed drafting, design, and execution. In most professions, this detailed work is hidden from view – who would want to follow the stress calculations for the structures of civil engineering. Even mathematicians conceal the detailed working of their proofs. In computing, we are fortunate that the greatest masters have devoted their energies to exposing the details,

in an elegant well-structured fashion, that makes such fascinating reading. I think we should expect and accept that most practitioners of the art of accurate programming will rightly keep the details to themselves.

The work that you have put into this beautiful study is further rewarded by more general lessons that may be learnt from it. Firstly, there is the skilful use of an invariant of the abstraction to code (by its violation) some essential aspect of the representation. This certainly suggests that the benefits of data abstraction are not going to be exhausted by construction of a library of standard representations.

A second lesson is addressed to the school of "reusers" of software. It is only your knowledge of an accurate specification of Dijkstra's binary search that has enabled you to reuse it.

I have long had a dream that the Computing Science Community would work together on a basic library of proven programs, gradually building it up over the years until someone might find a use for it. But hitherto, most work starts from scratch. By using both partition and bsearch, you show the way ahead.

But the pursuit of pure elegance is terribly beguiling. For example, I can't help thinking that you might have used a slightly more abstract representation of the array in your abstract algorithm: represent (b, u, v) as two sequences of sequences. Partition the first chunk from the second sequence. Move the first chunk from the second sequence to the first whenever it is small enough. The "sentinel" representation of the ends of the array is a bit sordid, and could be separated out.

But at best, this would be just a little more polish to a highly polished surface, more likely to cloud it than to improve it. An engineer and a scientist has to understand when it is best to let well alone. I hope you will find time and pleasure in adding to your collection of beautiful algorithms beautifully presented – in your next book perhaps?

Yours sincerely,

Tony.

Dear Don,

Thank you very much for your delightful contribution to my birthday book. It certainly illustrates my main goal in the design of notations — that they should have strong manipulative power, and that they should actually mean something. But your idea is even more beautiful than the notation you have chosen to express it: it is strongly symmetric in its two operands, so why imprison one of them in brackets? Maybe the single many operator [1] would serve most purposes?

But that's a real irrelevancy compared with the main messages of your paper — the immense power of generating functions and continuous mathematics in general. There seems to be nothing I can find in my

research, or in the other excellent articles in the book, that comes anywhere near matching that! "Related work and open problems" seem to be a lot less prominent in my articles, and those of my research colleagues. And that is a symptom perhaps of scientific immaturity, or worse. This may be the fate of all "foundationary" studies — logic, topology, category theory, semantics, ...

But it also explains why I find your work so admirable, in drawing together results from continuous and discrete mathematics, and developing them to the immense benefit of Computing Science.

Yours sincerely,

Tony.

Dear Rod,

Thank you for your letter of good wishes. Robin told me and Jill that Kaija has been very ill, and our deep sympathy goes out to you and your family. Our best wishes too for time to come, though presently clouded by sorrow.

Thank you too for your contribution to my birthday book. It has been a real pleasure to read your contribution as well as the others. The introduction has given me a real insight into the motivation for the use of both hidden sorts and institutions. The insight of identifying an object with a set of paths (and an alphabet of operations) seems to be the

same as led me to the trace model for communicating sequential processes. I always felt (following Robin's inspiration) that this must be some irreducible model of their behaviour: your constructions give a precise characterisation and foundation for that intuition. Am I right?

The notion of an institution perhaps also embodies my ideal of the close interlinking of models and algebra, and the essential role which they play separately and together. The Σ-morphisms are (families of) abstraction functions, showing how one model can be represented or implemented in another, and the Σ-morphisms are translations of notations, presumably intended to ensure that the abstraction function commutes with the operators of the target algebra, in the manner that I required in my "proof of correctness of data representations". He Jifeng is now using this paradigm in constructive derivations

of correct compilers for hardware.

You were my inspiration originally to undertake a study of category theory, and I have derived enormous benefit from. It was hard work to begin with; and after a year, all I seemed to have achieved was to make myself incomprehensible to my Fellow computer scientists, to category theorists, and to those even in the intersection of the two sets! I think my inclination is to concentrate on a single model at a time, and use ordinary mathematical methods to derive its algebraic properties. Then I am willing to specify an explicit abstraction functions between two models, and use it to prove a specific correctness result for a translator. In this way, I copy the run-of-the-mill mathematical modeller, who does not need to make explicit distinctions

between syntax and semantics, between validity and theoremhood. We are much more interested in "what are all algebraic laws satisfied by this mathematical model?" rather than "what are all models satisfying these laws" Institutions generalise these questions to classes of models and algebras, and institution morphisms go one stage further than I have been prepared to follow. - so far. If anything can persuade me onward, it is your article. Thank you again for it, and all your many insights and encouragement.

Yours sincerely

Tony

27 Jan 1994

Dear Cliff,

A second letter to thank you specially for your contribution to my birthday book. It tells me a lot about your recent work in linking the π-calculus with object-oriented programming. As you know, my own work on CSP was very much inspired by Simula 67, and it was especially helpful that your examples are already familiar to me: VAR is 4.2 (X7) in the CSP book, and Symtab is structurally similar to TREE. 4.5 X9. (pages 137 and 165). More general paradigms of dynamic object creation and linkage are contained in Chapter 6. — though I'm not satisfied with it.

The main difference in our approaches is that I was deliberately trying to maintain the kind of locality and scope control that we had in ALGOL W and PASCAL PLUS, and the hierarchical structure of qualified names that reflect it

in languages like SIMULA and COBOL. You actually start with the total freedom of machine-code reference passing, and have to work a bit harder to establish the validity of reasoning based on privacy.

A second difference is my determination to use the same language for describing programs and reasoning about them, at least at the algebraic level. Again, I was frightened by the extra level of indirection involved in translation.

A third difference is a blinding prejudice which I share with Dijkstra against operational reasoning. So I prefer to define the meaning of my notations at as high a level of abstraction as possible — preferably as observations in terms of which a user might prefer to specify the behaviour of the program before it is written. It is this that ensures that my calculus will be compositional at the design stage. Operational reasoning is more applicable after the program is written, e.g. for debugging purposes.

Of course, my approach uses a lot of standard mathematics — set theory, bound variables, fixed points ... The amazing promise of the π calculus is that all these features can be modelled operationally from the ground up, with nothing more than the "unique identifier" or "reference" concept. I remember a similar excitement when I first met Church's lambda calculus, in an article by Bohm on "the Curch." But it's a long way from that elusive "midpoint" between theory and practice, where I try to centre most of my work, if I can.

Your earlier work on shared-store parallelism is becoming more and more relevant to our hardware compilation proofs. It also seems to fit well with new developments in Jay Misra's thinking. All good things come together in the end.

Yours sincerely,

Tony

25 Jan 1994

Dear Bill,

Thank you again for your written contribution to my birthday book. It combines an excellent summary of the design philosophy which we shared in the research on CSP with a clear description of FDR, one of the most promising approaches to its practical application. And the exposition is exactly judged to tell me what I want to know about it. So my only comments are questions

Would it be reasonable to reference Hennessy and de Nicola's work on normal forms of CSP (in addition to [8])?

Would it be possible to apply similar techniques to CCS, and help to speed up the concurrency workbench?

(3) Would it help to compile low level processes to normal form – particularly if there are a lot of identical ones?

(4) Which are the eight coloured pegs in the game of solitaire (P-H. did not run to colour printing)?

(5) Solitaire is dual: any solution can be run backwards, interchanging pegs and holes. So the reservation of 8 pegs to the last ten moves is equivalent to requiring 8 specific holes after the first ten moves – an obviously drastic simplification.

(6) Solitaire is a monotonic game, in which earlier states never recur. Would it be worth while to allow an option to throw away earlier states, rather than merging them?

(7) These games would be nice examples for competition with BDDs, and other checkers.

(8) There is work on esctension of CTL model checking to time (including the duration calculus) in VERIMAG (Ahmed Bouajjani). They assume all time-specifications are linear, and use LCM.

(9) SDL seems to be the most popular specification language in the telecommunications industry. (much more than LOTOS).

(10) Could you generate specifications from a limited class of trace/refusal predicates? (MSc project?)

I couldn't quite assess (or even understand) all your proposals for improvement. From an academic point of view, the most interesting would be fuller exploitation of compositionality: commercially too, it may give you the biggest edge over rivals.

Yours sincerely

Tony

25 Jan 1994

Dear Samson,

Thank you very much for your contribution to my birthday book. I am particularly pleased at your quotation from my traces paper. I also agree strongly with your own analysis of current problems in the field of semantic research, and your own approach to their solution. I have always used elegance of algebra as a criterion in the design of a single model. You (rightly) propose category theory as a criterion of elegance for an algebra, which you intend to be applicable to a whole range of models.

I too am pursuing a kind of universal philosophy, under the slogan "Programs are Predicates." The objects are the alphabets of

permitted free variables. A predicate can be made into an arrow in your symmetric monoidal category by partitioning the alphabet into two sets, the "input" variables and the "output" variables. The symmetric monoidal axioms say nothing more or less than "this can be done in any way you like." And one of the ways that you may like will give you standard sequential composition.

One of the advantages of starting with predicates is that each hom-set is endowed with a natural ordering (implication), which is equally useful in modelling non-determinism in implementations and refinement of specifications, designs, and programs. By endowing the predicates with the structure of a complete lattice, I get recursion as well. I am suspicious of more subtle definitions of non-determinism, which may involve more clever reasoning and less efficient implementation (eg backtracking)

When reading your work, I have often felt that if I studied it more closely I would be able to understand it. A study of this short article reveals that some of my problems are due to the author rather than the reader – terms and notations used without definition, unexplained deviations from the referenced "standard". I am greatly looking forward to studying the fuller account, and discussing it with you.

Yours sincerely,

Tony

25 Jan 1994

Dear Jim,

Thank you very much again for your contribution to my birthday book. It is a beautifully stated description of what must be the essence of any solution to the problems of engineering of software — or anything else of such complexity. You may be interested in one Company I have visited which made a policy of throwing away all design documentation, on the grounds that it is too often inconsistent with the code, so its use is actually misleading.

I am now consulting for a Company (a different one) which has to maintain a twenty million line program. Here the problem is that a typical upgrade requires change to

to a dozen or so disjoint modules. They hope to discover which modules are relevant by means of an execution trace! This problem seems insoluble by your methods, or by any other. All we can say is that the whole structure of the original program has been ill designed for the kind of future updates that are actually needed. We should reiterate Dijkstra's advice that a program should be designed as the "family of programs that it may evolve to."

Perhaps the basic problem arises because our program modules are in one-one correspondence with the intervals in main store which hold the code which is compiled from them. Object orientation may help to break up this correspondence. But maybe we need a new concept of "procedure call"

as an event in which more than two objects can participate. This is an idea taken from the definition of parallel composition of CSP; but I don't think anyone yet has tried to put it into practice.

Keep up the good work!

Yours sincerely

Tony.

25 Jan 1994

Dear Robin,

I very much appreciate the kind Foreward that you contributed to my birthday collection. It has always been my goal to win your respect and appreciation, especially since we often take opposite (though never opposing) approaches to the same problem.

In the past, I have always felt that kind words place upon me an obligation to live up to them. At last I can feel more relaxed, and enjoy them all the more

Yours

Tony.

25 Jan 1994

Dear Carroll,

Many thanks for your excellent contribution to my birthday book. I have recently been enquiring after a definitive treatment of the relationship between relations and conjunctive predicate transformers; and now here it is.

And of course it is so nicely written, both in the details of the proof, and the overall structure of the development. It is like a detective story, with plenty of clues thrown in, challenging the reader to anticipate the writer. But the real pleasure and benefit is not just in the solution to the mystery but in the beautiful treatment of so many important topics along the way.

Thank you two for recalling the story from my first visit to Australia. I really do like questions from students, who are not in anything similar to flies, which I experienced on my second visit in good measure.

Yours sincerely

Tony

25 Jan 1994

Dear Michael,

Thank you very much for your excellent contribution to my birthday book. You are correct in your assessment of my appreciation of the broader canvas. And I agree strongly with the views that you so clearly and convincingly express. The link with reality is the most important thing; and only clear well-structured description can achieve that: notations, formalisms and calculation can start only after the link has been established.

I also strongly agree that a multiplicity of description methods must be brought to bear at different stages of the design, for different parts of the product, and for different classes of user. Like you, I deplore one aspect

of our current culture of software development, that managers, salesmen, politicians insist that all problems must be solved within a single framework; and too many researchers are willing (or coerced) to promise such a panacea. Where would mathematics be if all problems had to be posed and solved in graph theory? Or metallurgy, if molybdenum were the only metal in fashion? Until researchers are freed from the obligation to worship fashionable buzz-words, we will never be able to establish a rational structure for our discipline, with specialised notations and reasoning methods for different purposes, and with secure interfaces for combining them when necessary.

This is exactly where mathematical formalisation can make its most significant contribution; we can design embeddings which securely relate theories at different levels of abstraction; and we can clarify the restrictions and protocols by which different descriptions can be applied to different parts of a product. It is in the interfaces between technologies that errors are most frequent, most subtle, and most damaging. And that is not the only reason for theorists to work on interfaces. Such work appeals to my curiosity about the underlying structure of our subject, and is essential if it is ever to be taught effectively at University level.

So I am really pleased that my views agree so well with those that you have more laboriously acquired from a lifetime of work in close contact with "the real world". I certainly

propose to devote what efforts remain to me to the "clear descriptions of complex systems in which many domains meet and interact". But more significantly, I want to encourage others to take up the good work. The encouragement that you give in essays like this will be even more effective.

Yours sincerely

Tony

25 Jan 1994.

Dear Edsger,

Many thanks again for your very nice contribution to my birthday book. I am taking a leaf out of your book (metaphorically, of course) in reading the articles and writing to the authors.

What a lot we have learnt from the relational calculus! I have just been rereading your last Marktoberdorf lectures as well. I have been hoping, with the help of Burghardt von Karger, to find a family grouping that would include more than three calculi.* Your design of calculational proofs is full of good ideas and good advice. But it didn't prevent me spending

---

* including ones without [illegible]

all day today on a wild goose chase!

My publisher organised a very nice party for me on the 12th, with a fair number of contributors as well as members of the Department. There are now 85 books in the series, so there is a good hope that I'll get to 100 books, before retiring.

Then there's Marktoberdorf. I hope to get down to planning my lectures straight away. I hope we remember this time to elect Richard Bird as Director. See you there.

We very much enjoy the quilt, which hangs in pride of place. Thank you, Ria, and also Ann, and best wishes for her 80th. We're planning for my mother's 85th.

Yours

Irene

25 Jan 1994

Dear Jay,

Thank you again for your beautiful contribution to my birthday book, and the kind dedication. I find your message totally convincing: this must be the right way to reason about highly parallel networks; and your exposition must be the right way of explaining it. The absence of pictures is most illuminating.

And the achievement is amazing: I have myself tried to explain "Fast Fourier Transform Free from tears" Computer Journal 20(1) 78-83 (Feb 77). Kruseman Aretz had done it better. I have read Dijkstra on Batcher's Baffler, and still been baffled. You have solved both puzzles, merely as illustrations of the attractiveness of your reasoning methods.

better not to start a program that it is going to take too long to finish!

The explicit mention of time makes an immediate entry to considerations of complexity. If you want to abstract from that, it is always possible to precede each program by an arbitrary selection of machine speed ε.

$$\varepsilon :\in \{\varepsilon \mid \varepsilon > 0\};\ \text{program}$$

I think this restores the theorist's "full abstraction" by reducing termination to a single bit.

There is another brilliant idea in your paper, that is the achievement of recursion without using a formal identifier for the recursive call! Indeed, it avoids semantic complication! It's also a very practical way of getting programmers to write assertions just where they are most needed! It really

achieves Floyd's original goal of Assigning Meanings to Programs.* Could you also solve the problem of first order parameter passing simply by allowing substitutions in the assertions which stand for the recursive call? If the compiler can recognise the ~~a~~substitution by pattern matching, generation of parameter passing by copy-in, copy-out could be automatic. But I don't think this will work for higher order procedures, where positional addressing of (a few) parameters is convenient.

I feel you have removed one of my mental blocks. I was always afraid of ~~e~~ recursive equations of the form

$$X \Leftarrow F.X$$

because they lead to strangest fixed points, miracles, and partial correctness, of which I have long been ashamed. But your explicit

---

*

introduction of t immediately leads to unique fixed points and total correctness, removing grounds for both fear and shame.

So I have much to thank you for: I hope I get as much reward from the other articles in the book, and that we meet again soon.

Yours sincerely

Tony

OXFORD UNIVERSITY COMPUTING LABORATORY
Programming Research Group

James Martin Professor of Computing: C. A. R. Hoare FRS
Professor of Computing Science J. A. Goguen

Tel: +44 865 273840
Tel: +44 865 283504

14 Jan

Dear Bill,

How can I ever thank you enough for your organisation of my Birthday Book — quite the best present I could ever imagine? I was immensely pleased when you told me about the project on my last birthday; but I had no idea how well it was going to turn out. Everybody seems to have set out to write something that I would want to read, and be able to understand and enjoy. And of course the prize was a complete surprise, and an honour which I greatly appreciate.

Wolfson Building
Parks Road

Enquiries: +44 865 273838
FAX: +44 865 273839

Email: *carh*@comlab.ox.ac.uk
goguen@comlab.ox.ac.uk

I hope you know (because I don't very often tell) how much inspiration I have drawn from your work in the development of my research approach and philosophy, and in the academic development of computing at Oxford. I certainly know how many interesting and useful things you must have put aside to edit the volume. Gratitude and esteem are but weak words to express what I feel.

Yours sincerely

Tony

OXFORD UNIVERSITY COMPUTING LABORATORY
Programming Research Group

James Martin Professor of Computing: C. A. R. Hoare FRS
Professor of Computing Science J. A. Goguen

Tel: +44 865 273840
Tel: +44 865 283504

Jan 14

Dear Jim,

I have sent letters of thanks to all the official contributors to my "Birthday Book", which is the best birthday present one could imagine. But the first thing that strikes the eye is the quality of the text, which fully lives up to the luxury of the binding. For that I want to thank you especially for the care and skill and (in places I am sure) even understanding which you have expended on it. It was so much more than could be deserved or repaid – but I hope

Wolfson Building
Parks Road

Enquiries: +44 865 273838
FAX: +44 865 273839

Email: carlh@comlab.ox.ac.uk
goguen@comlab.ox.ac.uk

to have the chance to try.

I hope things are going well for you at Reading, and that you are still as cheerful as you look!

Yours sincerely

Tony

OXFORD UNIVERSITY COMPUTING LABORATORY
Programming Research Group

James Martin Professor of Computing: C. A. R. Hoare FRS
Professor of Computing Science J. A. Goguen

Tel: +44 865 273840
Tel: +44 865 283504

Jan 14

Dear Jim,

I have sent letters of thanks to all the official contributors to my "Birthday Book", which is the best birthday present one could imagine. But the first thing that strikes the eye is the quality of the text, which fully lives up to the luxury of the binding. For that I want to thank you especially for the care and skill and (in places I am sure) even understanding which you have expended on it. It was so much more than could be deserved or repaid - but I hope

Wolfson Building
Parks Road

Enquiries: +44 865 273838

Email: carh@comlab.ox.ac.uk

to have the chance to try.

I hope things are going well for you at Reading, and that you are still as cheerful as you look!

Yours sincerely

Tony

OXFORD UNIVERSITY COMPUTING LABORATORY
Programming Research Group

James Martin Professor of Computing: C. A. R. Hoare FRS — Tel: +44 865 273840
Professor of Computing Science J. A. Goguen — Tel: +44 865 283504

Dr M J C Gordon
Department of Computer Science
University of Cambridge
Corn Exchange Street
Cambridge

14 January 1994

Dear Mike,

Thank you very much for your good wishes for my Birthday and your contribution to the book of essays in my honour. That was a marvellous birthday present: not just any old book bought in a shop, but one which no-one had ever seen before, one which you have written specially for me.

As a delightful present for my last birthday, Bill Roscoe told me of your plans: and now I have seen how well they have turned out (in a beautiful presentation copy) my delight is unbounded. I shall greatly enjoy reading your contribution, so that I may enjoy the benefits of it at my next birthday, and continue to enjoy them for the many happy returns that you have wished me.

Yours sincerely,

Tony

Wolfson Building
Parks Road

Enquiries: +44 865 273838
FAX: +44 865 273839

Email: *carh*@comlab.ox.ac.uk
goguen@comlab.ox.ac.uk