CONCUR Test-of-Time Award for the Period 1994–97
Interview with Uwe Nestmann and Benjamin C. Pierce

Adam D. Barwell, Nobuko Yoshida

Imperial College London, UK

Francisco Ferreira

Royal Holloway, University of London and
Imperial College London, UK

Abstract

Last year, the CONCUR conference series inaugurated its Test-of-Time Award, the purpose of which is to recognise important achievements in Concurrency Theory that were published at the conference and have stood the test of time. This year, *Decoding Choice Encodings* by Uwe Nestmann and Benjamin C. Pierce was one of four papers chosen to receive the CONCUR Test-of-Time Award for the periods 1994–1997 and 1996–1999 by a jury consisting of Rob van Glabbeek (chair), Luca de Alfaro, Nathalie Bertrand, Catuscia Palamidessi, and Nobuko Yoshida. This article is devoted to the engaging and interesting interview conducted with Uwe Nestmann and Benjamin C. Pierce via video conference.

Keywords:
Pi-Calculus, Encodings, Lambda-Calculus, Distributed Systems, Concurrent Systems, Interview

---

1. Introduction

Four papers were awarded CONCUR's Test-of-Time Award at this year's conference[1]. The award, first issued in 2020, aims to recognise important

---

[1] Held online between 24 August 2021 and 27 August 2021. The other recipients of the award were: Janin and Walukiewicz [1]; Bouajjani, Esparza, and Maler [2], and Alur, Hen-

Preprint submitted to Journal of Logical and Algebraic Methods in Programming November 15, 2021

achievements in concurrency theory that have stood the test of time since their publication at the CONCUR conference.

Nestmann and Pierce's 1996 paper, *Decoding Choice Encodings* [7], was recognised with the aforementioned award for making major strides in the study of the expressiveness of process calculi. It shows that, in a completely distributed and asynchronous setting, input-guarded choice can be simulated by parallel composition. More precisely, the paper constructs a fully distributed and divergence-free encoding from the input-choice π-calculus into the asynchronous π-calculus. The correctness of this encoding is demonstrated by establishing a semantic equivalence between a process and its encoding, thereby satisfying and strengthening the common quality criterion of full abstraction. As semantic equivalence it employs the asynchronous version of coupled simulation, and illuminates the surprising versatility of this notion by showing how it avoids the introduction of divergence in the encoding. This work formalizes ideas stemming from the programming language PICT, and has been very influential in the area of expressiveness in concurrency.

The study of the relative expressiveness of π-calculi began via the introduction of the asynchronous π-calculus by Honda and Tokoro [4], and in subsequent work by Boudol [5]. The asynchronous π-calculus was presented as a subset of the original synchronous π-calculus [6], and Nestmann and Pierce's paper provides a compelling answer to the question of expressiveness of the family of π-calculi. Nestmann and Pierce's work provides, for example, a positive result following the negative result presented by Palamidessi, which shows the impossibility of translating from the π-calculus with mixed choice into the π-calculus without mixed choice [8, 9]. Furthermore, the work by Nestmann and Pierce led to the first EXPRESS workshop [10] in 1997 that continues to explore this topic today.

---

zinger, Kupferman, and Vardi [3].

2. Interview

Nobuko: Congratulations on receiving the CONCUR 2021 Test-of-Time Award for your 1996 paper *Decoding Choice Encodings* [7]. Could you tell us briefly what lead you to embark on studying the expressiveness of choice in the asynchronous π-calculus?

Uwe: I built a typed λ-calculus with communication for my diploma thesis in 1991. It was capable of typing the Y-combinator, and I presented it at the Concurrency Club[2] at the University of Edinburgh. The Club asked me who my supervisor was, but I didn't have one at that time, being mostly self-driven. They advised me to get a supervisor first, and then look for a topic. I found Benjamin, who had this wonderful project at the time on trying to make a programming language out of the π-calculus (i.e. the PICT language [11, 12]). Choice encodings (or at least choice operators) played a role in PICT. He invited me to visit him in Paris.

Benjamin: I was in Paris at the time as part of a "nested postdoc." I did three postdocs after finishing my PhD at Carnegie Mellon University: one at the University of Edinburgh[3], one at INRIA-Roquencourt in Paris[4], and one at the University of Cambridge[5]. My time in Paris occurred during a leave of absence from Edinburgh.

Uwe: I was in Paris for one week, and Benjamin told me to try programming in his new language, PICT. I tried to write down the dining philosophers problem [13], in such a way that a philosopher can pick up a fork from either side. More precisely, given some process definition

$$\mathsf{Phil}_{\mathsf{det}}(f_1, f_2) = f_1?x.f_2?y.P$$

that represents a philosopher who deterministically picks up a fork one after the

---

[2] A group of 15–30 people, then run by Perdita Stevens and Julian Bradfield.

[3] Between January 1992 and December 1994.

[4] Between September 1992 and May 1993

[5] Between January 1995 and August 1996.

other, I instead wanted to write

$$\mathsf{Phil}(\mathsf{left}, \mathsf{right}) = \mathsf{Phil}_{\mathsf{det}}(\mathsf{left}, \mathsf{right}) + \mathsf{Phil}_{\mathsf{det}}(\mathsf{right}, \mathsf{left})$$

which represents a philosopher picking up forks non-deterministically in either order. Unfortunately, PICT did not allow this.

This interplay between choice and abstraction (and instantiation) was the start of it all from my point of view. I wrote up an exposé and I ended up actually working on just a third of that for my PhD thesis. Of course, at the time, there were technical reasons for Benjamin and Dave Turner being interested in choice constructs.

Benjamin: Besides Robin Milner, Dave Turner is, of course, the most important name that needs to be mentioned here. All of this was happening under the umbrella of Robin's wonderful work on the π-calculus and the amazing group that he had assembled at the time. He had this incredible set of students, including Dave Turner, Davide Sangiorgi, and Peter Sewell, doing all sorts of things with π-calculus. Dave, besides being a first-class hacker, was also a really good theoretician. He truly married the two. He and I started talking at some point about what kind of programming language you would get if you treated the π-calculus in the same way that the Lisp people treated the λ-calculus. What that led to was a lot of different language designs based on different versions of the π-calculus, but we kept wanting to make it simpler and simpler. Partly because we were thinking of it as possibly even a distributed language, not just a concurrent language. As everybody knows, the choice operator – in the full-blown π-calculus or CCS sense – is not a real thing in distributed systems: it's not implementable. So we were trying to make the underlying calculus simpler and simpler, and eventually wound up with this programming language with no choice operators at all. But, as Uwe discovered, there are things that you might want to do where choice is the natural primitive, such as the dining philosophers problem, which raises the question of how much of it can you get just by programming on top of plain parallel composition plus messages on channels. We found that programming a restricted form of choice was a little tricky.

However, what was *really* tricky was justifying that it was correct. The reason why it turned into a whole dissertation for Uwe was because the well-known notions of correctness that were lying around (e.g. full abstraction with respect to standard weak bisimilarities) did not apply to this situation. I remember being totally astonished at the length and technicality of the final proof that Uwe ended up doing.

Nobuko: Did you imagine at the time that your award-winning paper would have so much impact on the area of expressiveness in concurrency theory, and how do you feel now?

Benjamin: Maybe Uwe did; I did not. I think we were just following our noses.

Uwe: I would say both "yes" and "no". When it came to the CONCUR acceptance, I got the impression that we just about made it because the competition was so tough and the π-calculus was really popular at that time. There were six or seven π-calculus papers accepted at the conference; I don't know how many were in the submission pool. The tiny "yes" that I would like to say is because Kohei Honda foresaw it. When I gave the presentation at the Newton Institute just in the autumn of 1995 – that was the workshop that Benjamin organised on concurrent high-level languages[6] – Kohei came to me after the talk and said something like, "*maybe you don't know yet, but you will be known for this*". I can't remember the exact wording, but I think he called it *Nestmann's Theorem.* It was my first time in front of this crowd of experts and then he tells me, a PhD student, something like that. I didn't believe him, of course.

Benjamin: Kohei was ahead of his time in so many ways.

Nobuko: Could you tell us what the research environment was like in Edinburgh, and the UK as a whole, at that time and how it has influenced the rest

---

[6] The High-level Concurrent Languages: Foundations and Verification Techniques (HLCL) workshop was held between 2 October and 4 October 1995. It was organised by Benjamin C. Pierce and Matthew Hennessy.

of your career?

Benjamin: I arrived as a postdoc in Robin Milner's group. I was his last postdoc whilst he was at the University of Edinburgh, and then travelled with him to the University of Cambridge, where Peter Sewell and I were his first postdocs. I would say that both Edinburgh and Cambridge at the time were just incredible, and still are. At Edinburgh, you had Robin Milner, Gordon Plotkin, Don Sannella, Rod Burstall, Colin Sterling, and Randy Pollack. You also had students around you like Martin Hofmann, Philippa Gardner, and Marcelo Fiore. The list goes on and on. It was just an incredible place. People talked about amazing, deep, mind-bending things all the time. It was particularly an amazing place for thinking about concurrency. There were a lot people breaking new ground.

Nobuko: Benjamin, how did that experience influence your current research?

Benjamin: For one thing, it solidified my interest in language design. The whole PICT experience was so fruitful. It was so much fun working with Dave Turner on implementing this interesting language. Both the design and programming that we did as part of PICT gave rise to so many interesting questions. For example, it led us to think a lot about type systems for concurrency, and I can see echoes of those ideas in the work that you, Nobuko, and colleagues have done more recently with session types. Although I don't consider myself a core concurrency researcher any more, the experience gave me an appreciation for the theory of concurrency that draws me back to the area time and time again.

Nobuko: Uwe, how did it influence your research?

Uwe: I did my PhD at the University of Erlangen-Nürnberg, which was not so known at that time for theory, especially not for concurrency theory. I had the opportunity by a bilateral travel exchange programme[7] between these

---

[7] The travel exchange programme in question was called the Academic Research Collabora-

two universities pushed by my other supervisor, Terry Stroup, at that time. When I visited Edinburgh, not only was there so much competence around, but there was so much openness for any kind of idea. So much curiosity and joy. I was very lucky that I could visit the LFCS for a few days every couple of months. There, I was filled up with content and ideas. I also did a presentation in the π Club in Robin Milner's tiny office, with almost ten people sitting around a tiny blackboard, listening to my ideas and my problems. It was just unbelievable at this time. That kind of culture and atmosphere was so great. In May or June 1995, since we're talking about this particular paper, it was culminating in the crucial part where I was just before proving choice encodings correct. I only needed two ingredients. One came a week later by Davide Sangiorgi posting, for the first time, a short note on asynchronous bisimilarity (that eventually became [14]). The other was that we were rediscovering the notion of coupled similarity, mostly together in the π Club with Ole-Høgh Jensen and Robin Milner. Both Ole and Robin had different ideas and came to the same conclusion. I went back to Erlangen and found the old paper on coupled similarity [15] by Joachim Parrow and Peter Sjödin and, within a week, all of the pieces were mostly in place. I simply needed to write down the details and convince myself that it was correct. That was the crucial moment, and without Edinburgh, its culture, its openness, and the possibilities that it presented, the paper would not have happened, and maybe I would not even have become a professor at the Technische Universität Berlin. All because of this tiny situation and the congregation of bright people.

Nobuko: Studying expressiveness this way was quite new at that time, so you probably cared a lot about presentation and how to communicate your ideas. Do you have any comments about this aspect? I found that your paper remains very readable and very clearly written for such a subtle piece of work.

---

tion (ARC) and funded by both the British Council (BC) and the German Academic Research Council (DAAD).

How did you go about writing with this in mind? Aside from technical details.

Uwe: I was a great fan of Benjamin's presentation and communication skills at that time. I saw him on stage and read his papers, and I had the opportunity to interact with, and learn from, this impressive guy. I recently heard an aphorism that summarises what I learnt back then in trying to write this paper: *"Do not try to write such that you are understood. Try to write such that you cannot be misunderstood."* It's often underestimated how important the role of good notation is for getting things across. The same goes for graphical presentations. And then, polishing, polishing, polishing, polishing. *"Get simpler sentences,"* Benjamin always said. I'm German, you know, we like complicated constructions which are deeply nested, but I learnt to get it as simple as possible. Presentations were another thing. I found my presentation from the 1996 CONCUR conference, which had its table of contents written in the form **ABCDE**. Each letter was an initial of the concepts that I presented: **A**synchronous Choice (setting and encoding), **B**y Simulation (formulating correctness notion), **C**oupled Simulation (getting it right. . .), **D**ecoding Encodings (for establishing simulations), and **E**nd (conclusion and further work). I like playing with words and I admire the power and joy of well-chosen language.

Nobuko: I do remember your presentation. You highlighted coupled simulation as a part of Rob van Glabbeek's famous diagram [16, 17].

Benjamin: I have always cared a lot about good writing. Communicating ideas is really one of the most important parts of an academic's job. So it feels important to acknowledge the people I learned about writing from. The first was Don Knuth – his level of attention to writing, among the many other things he did, is very inspiring for me. The other was John Reynolds, who was one of my two supervisors as a PhD student, my other supervisor being Robert Harper. John Reynolds is the most careful writer that I have ever worked closely with. He once gave me a draft of one of his papers to proofread, so I started reading it, and I couldn't find anything to improve. That experience was both an inspiration and a humbling lesson to me.

The biggest thing I've learned over the years about writing is that the biggest ingredient of good writing is exactly what Uwe brought to this paper: the willingness to iterate until it's good. Good writers are people that stop polishing later than bad writers.

Nobuko: How much of your later work has built on your award-winning paper? What follow-up result of yours are you most proud of and why?

Uwe: I would like to mention three. Funnily, none of them were in the decade following the CONCUR paper. The reason may be because I was dragged into other projects, which were focussed on security protocols, π-calculus, and object calculi [18, 19]. By accident, I got back in contact with Ursula Goltz, who was one of my PhD referees: she was working on a project about synchronous and asynchronous systems. She asked me for literature because she knew I was digging deep in the 1980s about results on the first CSP implementations. Over the course of this project, I managed to directly build on my PhD work. I also found Kirstin Peters, who was a PhD student at the time, and who became interested in the same work. We found a number of remarkable observations having to do with distributed implementability and notions of distributability and what this may have to do with encodings between calculi. We discovered a hierarchy of calculi, where you can very easily see which of them are at the same level of distributed implementability. We found that the asynchronous π-calculus, like many others, is actually not fully implementable in a distributed system. There is the ESOP paper in 2013 [20], which I'm very proud of. Kirstin pushed this research much further.

Another follow-up work concerns the notion of correctness that we were applying in the awarded paper. The work was primarily about a direct comparison between terms and their translations. Not by plain full abstraction on two different levels and having an if-and-only-if, but a direct translation so you could not distinguish a term from its translation. This kind of observation led to a reevaluation of the research on what we actually want from an encoding. What is a good criterion for a good encoding? This culminated in the work with Daniele Gorla, where we criticised the notion of full abstraction in the sense that, whilst it's a very important notion, you can easily misuse it and get to wrong, or useless, results. (We also emphasized the importance of operational correspondence, and Daniele went on to establish his, by now, quite standard and established set of criteria for what makes a good encoding [21].) That is a nice highly abstract paper with Daniele in Mathematical Structures in Computer Science in 2016 [22]. So also well, well after the CONCUR paper in 1996.

Within the last two or three years, my PhD student, Benjamin Bisping, studied algorithms and implementations for checking coupled similarity [23]. We found an amazing wealth of new views on these kinds of equivalences that are slightly weaker than weak bisimilarity. (Like Kirstin Peters and Rob van Glabbeek who further showed that coupled similarity is in fact very closely connected to encodings, in general [24].) So back to the roots, in a sense, to what we were doing 25 years ago. Seeing these developments is a lot of fun.

We also published the survey article *Coupled Similarity – The First 32 Years*, for the Festschrift for Robert van Glabbeek [25]. It's basically an advertising paper for this great notion of equivalence, which is highly underestimated. It is, in a sense, much better than weak bisimilarity. Especially if you're interested in – and this is my favourite domain – distribution, distributability, and distributed implementations.

Nobuko: Benjamin, do you have any further comments?

Benjamin: The answer is a little more oblique for me. Besides the awarded paper, I haven't written papers about choice encodings, and things like it. What it did for me, however, was to really solidify my interest in the asynchronous π-calculus as a foundation for programming languages – and as a foundation for thinking about concurrency – because the awarded paper, Uwe's result, teaches us that the asynchronous π-calculus is more powerful than it looks – powerful enough to do a lot of programming in. It brings to mind the famous quote attributed to Einstein, *"Make everything as simple as possible, but no simpler."* I felt like the asynchronous π-calculus was kind of "it" after seeing this result.

That calculus then became the foundation for a lot of my later work on language design and type systems for concurrency.

Uwe: The encodings we did back then went into what is now called the *localised asynchronous π-calculus* [26], but it simply wasn't known back then. The localised asynchronous π-calculus is at a perfect level of distributed implementability, as we now know.

Nobuko: This is partly also work that Massimo Merro did with Davide Sangiorgi [27], right?

Uwe: Yes, they did this few years later, towards the end of the 1990s.

Nobuko: What uses of the notion and technique you developed in the awarded paper have you found in the literature that you found unexpected? What kind of application in other areas, such as programming languages, are there in general?

Uwe: It was unexpected that the asynchronous π-calculus would be this foundational model. However, as I said earlier, it turned out that it is the *localised* asynchronous π-calculus that is really the foundation for this kind of implementability. It would be interesting to check, ultimately, how much of the design of PICT is based on the localised asynchronous π-calculus. The idea of the calculus is basically: you cannot receive on received names. You can only send on them, or pass them on.

Benjamin: When you receive a name, you can't receive on it?

Uwe: You can only use a name you've received to send messages on, or to pass it on as an object. The point is that this is exactly what you get by syntax from the join calculus [28], which is the version that was done for distributed implementation. It's also the same principle that is behind the Actor model [29]. In the Actor model, you can never receive on received names, you can just send to actors, who have mail boxes, and they essentially run local input-guarded choice. These all reside on the same level in our hierarchy. There are very simple encodings between the Actor model (there is an Actor π-calculus by Agha and Thati [30]), the Join calculus, and the localised π-calculus. Moreover, there are distributability-preserving encodings between them. Thus they live at the same level. Conversely, the asynchronous π-calculus, i.e. without this locality principle, is not on the same level.

Benjamin: Why?

Uwe: Think about a distributed system. You need to route messages when you send them to participants. If there are many receivers sitting on different locations, you need to decide which one to route the message to. Maybe those locations are waiting on messages right now, or maybe not, but in essence you run a distributed consensus to find out which mailbox the message needs to go in. Here, the locality principle of actors, and join, and the localised π-calculus, to some extent, fixes the location of receivers, making the job of routing messages much simpler.

Benjamin: So, the reason why that wouldn't work is that, ultimately, you have to agree on where the receiver is. Indeed, also the fact that the receiver exists. If you know for certain that a receiver exists, then that's probably equivalent to knowing where it is, but agreeing on that fact might be hard.

Uwe: The consequences of an extension of that with fault tolerance. Or faults, and then tolerance.

Benjamin: But if you don't go that far, is there a theorem that says you cannot implement the asynchronous π-calculus in a distributed way?

Uwe: I was talking about this hierarchy that we had in the ESOP paper [20]. There are three levels, and there are two synchronisation patterns that make the difference between these levels. The level that distinguishes the localised π-calculus from the asynchronous π-calculus is what is called an *M-structure* [31, 20]. It's known from the Petri net area, that's why it was rediscovered with Ursula Goltz, and we found it in process calculi as well. Intuitively, the M-structure says: you have two independent actions that could be implemented on different (i.e. distributed) locations but if there is a third action that depends on resources that are shared with the other two, then they must all be implemented on the same location. As with an "M", you have the "heads" on the top, they are the resources that you need. The legs on the two sides are independent, but there is an inner "leg" connecting the others. That is, in essence, the thinking in Petri nets. We have reformulated the M-pattern of Petri nets in terms of labelled transition systems in order to make it somewhat model-independent. As a result, we may then look for the occurrence of M-structures also within process calculi. This then amounts to looking for process expressions whose transition systems contain M-structures. We can reproduce these kinds of M-structures in the asynchronous π-calculus, but not in the localised π-calculus, the actor π-calculus, or the join calculus. And then we get to the other level in our hierarchy, which is where you find the mixed choice π-calculus, amongst others. There is another synchronisation pattern that makes a distinction between the level with the mixed choice π-calculus and the level with the asynchronous π-calculus. This is what we call a ★ pattern [20]. Intuitively, it can be thought of like the dining philosophers with at least five people. You need an odd number of participants, that can form two Ms, which you can put together in a circle. You then have a very simple criterion for distinguishing between these levels. As you can see, I'm very enthused about this paper, but it's effectively a consequence of the awarded paper, only twenty years later. It plays on the same theme, and facilitates understanding more about distributed implementations.

Nobuko: What do you think of the current state and future directions of the study of expressiveness in process calculi and, more generally, concurrency theory as a whole?

Uwe: Back then, in Cambridge, I had many discussions with Peter Sewell. At the time, we joked by saying, *"now we know how to do process calculi, we can do five of them for breakfast."* We know the techniques, we know how to write down the rules, we know what to look for in order to make it good. I would say that for studying encodings nowadays it's at approximately the same level of maturity: we know what to look for when writing down encodings and the pitfalls to avoid. What I found most interesting today is that, often enough, the proximity between encodings and actual implementations is very close. This may be because the programming languages that we can use are much more mature. We can use convenient abstractions in order to more-or-less straightforwardly write down encodings.

Regarding the current state and future directions, the EXPRESS/SOS workshop [32] still exists. It attracts great papers. I think we had an impact on concurrent programming. For example, if you look at the Go programming language [33, 34], the concurrency primitives that you find are essentially a process calculus. It features message passing, choice, and even mixed choice.

I cannot say right now that there are deep, deep, deep questions to be solved about encodings except for finding out what Robert van Glabbeek's criteria [24] have to do with Daniele Gorla's criteria [21]. There is an ongoing debate, but the issues are quite technical. What could use more research is typed languages, typed calculi, and typed encodings. It has been done, and we have many nice results, but I think there are still some open questions on what the ideal criteria should be for those.

Nobuko: What advice would you give a young researcher interested in working on concurrency theory and process calculi today?

Benjamin: My best advice for people that want to do theory is: keep one foot in practice. Don't stop building things. That's the way you find interesting problems. It's the way you keep yourself grounded. It's the way you make sure that the directions in which you're looking and the questions that you're asking have something to do with real systems. It's the way to stay connected to reality whilst also generating great questions.

Uwe: Having a foot in practice is also good for checking and finding mistakes in your reasoning. Apart from that, I would not like to push for any particular area for concurrency theory. Instead, my advice is to get the best possible supervisor that you can find and then work on his project. This is very general advice but be patient, dig deep, and never give up. It took me two years until the pieces fell together in one week. So be patient, dig deep, train your communication skills, and practice networking. What I found very useful for my own career was to learn the basics and the history of your field. Understand what has already been found, and what that means even twenty years after publication. I learned a lot from the early 1980s papers on first implementations of the communication primitives of CSP. There is one supposedly deadlock-free implementation of the generalized alternative command algorithm [35], which was discovered to be incorrect fourteen years later; it was not actually deadlock free [36]. So, in conclusion, work on hard problems, dig deep, be patient, and communicate well. This is also the best way to get help.

Nobuko: This is the last question: what are the research topics that currently excite you most?

Benjamin: I will name two. One is machine-checked proofs about real software. Over the past fifteen or twenty years, the capabilities of proof assistants, and the community around them, have reached the point where you can use them to verify interesting properties of real software. This is an amazing opportunity that we are just beginning to exploit.

On a more pragmatic level, I'm very interested lately in testing. Specifically, specification-based (or property-based) testing in the style popularised by QuickCheck [37]. It's a beautiful compromise between rigour and accessibility. Compared to the effort of fully verifying a mathematically stated property, it is both incredibly easier and lower-cost. Yet, you can get tremendous benefit from both the process of thinking about the specification in the mathematical way that we're used to in this community, and from the process of testing against, for example, randomly generated or enumerated examples. It's a sweet spot in the space of approaches to software quality.

Nobuko: These things are still very difficult for concurrency and distributed systems. Do you have any thoughts on this, because proof assistants for concurrency theory are, I think, still quite difficult compared to hand-written proof?

Benjamin: Yes, in both verification and testing, concurrency is still hard. I don't have a deep insight into why it is hard in the verification domain, beyond the obvious difficulty that the properties you want are subtle. However, in the testing domain, the reason is clear: the properties have too many quantifier alternations, which is hard for testing. Not impossible – not always impossible, anyway – but it raises hard challenges.

Uwe: There's a recurring pattern in what I like doing and that is always to do with looking at different levels of abstractions. You can think of it in terms of encodings or as a distributed system, and I was always wondering about the relation between global (higher-level) properties and local (lower-level) implementation of systems. Applying formal methods, formal models, and theories at this problem has always been what I've liked. I still do that, albeit more on fault-tolerant distributed algorithms. At best, doing mechanical verification of those. Mechanical verification is still hard and you can easily put PhD students into a miserable state by dragging them onto a problem that takes an awful lot of time, and then you get out one paper, with the proof in Isabelle (in our case). On the other hand, it's increasingly a tool that we just use. The more you've done, using a proof assistant, the more you integrate it into your everyday life. Some students, as a standard, test their definitions and their theorems and do their proofs in Isabelle and we now even have undergraduate students using that. Bright ones, of course, but it's increasingly becoming quotidian. Recently, we have also been interested in understanding how people learn how to do proofs. It's a long, difficult, mental process and there are a number of theories about how this actually works, and whether this works. Furthermore, what is the impact of using proof assistants for learning how to do proofs? Does it actually help? Or does it actually hinder?

Benjamin: Anecdotally, it would appear to turn people into hackers.

Uwe: We're talking about computer science students, not maths students. Programming is proving, proving is programming. This is of course a slogan from type theory, but one may actually use it as a motivation to write down first proofs, getting feedback from the proof assistant, and go from there. This is something we're interested in, in actually understanding this process of learning how to do proofs.

Nobuko: Thank you both very much for giving us your time.

Acknowledgements

We thank Uwe Nestmann and Benjamin C. Pierce for their time and assistance in the production of this interview. This work was supported by EPSRC grants EP/T006544/1, EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/1, EP/N028201/1, EP/T006544/1, EP/T014709/1, EP/V000462/1, and NCSS/EPSRC VeTSS.

References

[1] D. Janin, I. Walukiewicz, On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic, in: U. Montanari, V. Sassone (Eds.), CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings, Vol. 1119 of Lecture Notes in Computer Science, Springer, 1996, pp. 263–277. doi:10.1007/3-540-61604-7_60.

[2] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: Application to model-checking, in: A. W. Mazurkiewicz, J. Winkowski (Eds.), CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings, Vol. 1243 of Lecture Notes in Computer Science, Springer, 1997, pp. 135–150. doi:10.1007/3-540-63141-0_10.

[3] R. Alur, T. A. Henzinger, O. Kupferman, M. Y. Vardi, Alternating refinement relations, in: D. Sangiorgi, R. de Simone (Eds.), CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings, Vol. 1466 of Lecture Notes in Computer Science, Springer, 1998, pp. 163–178. doi:10.1007/BFb0055622.

[4] K. Honda, M. Tokoro, An object calculus for asynchronous communication, in: P. America (Ed.), ECOOP'91 European Conference on Object-Oriented Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1991, pp. 133–147.

[5] G. Boudol, Asynchrony and the Pi-calculus, Research Report RR-1702, INRIA (1992).
URL `https://hal.inria.fr/inria-00076939`

[6] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, I, Information and Computation 100 (1) (1992) 1–40. doi:10.1016/0890-5401(92)90008-4.

[7] U. Nestmann, B. C. Pierce, Decoding choice encodings, in: U. Montanari, V. Sassone (Eds.), CONCUR '96: Concurrency Theory, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 179–194.

[8] C. Palamidessi, Comparing the expressive power of the synchronous and the asynchronous π-calculus, in: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '97, Association for Computing Machinery, New York, NY, USA, 1997, p. 256–265. doi:10.1145/263699.263731.

[9] C. Palamidessi, Comparing the expressive power of the synchronous and asynchronous π-calculi, Mathematical Structures in Computer Science 13 (5) (2003) 685–719. doi:10.1017/S0960129503004043.

[10] C. Palamidessi, J. Parrow (Eds.), International Workshop on Expressiveness in Concurrency, EXPRESS 1997, Vol. 7 of Electronic Notes in Theoretical Computer Science, Elsevier B.V., Santa Margherita Ligure, Italy, 1997.

[11] B. C. Pierce, D. N. Turner, Pict language definition, Tech. rep. (1997).
URL `https://www.cis.upenn.edu/ bcpierce/papers/pict/pict-4.1/Doc/defn.ps.gz`

[12] B. C. Pierce, D. N. Turner, Proof, Language, and Interaction: Essays in Honour of Robin Milner, 2000, Ch. Pict: A Programming Language Based on the Pi-Calculus, pp. 455–494.

[13] C. A. R. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677. doi:10.1145/359576.359585.

[14] R. M. Amadio, I. Castellani, D. Sangiorgi, On bisimulations for the asynchronous π-calculus, in: U. Montanari, V. Sassone (Eds.), CONCUR '96: Concurrency Theory, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 147–162.

[15] J. Parrow, P. Sjödin, Multiway synchronization verified with coupled simulation, in: W. Cleaveland (Ed.), CONCUR '92, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 518–533.

[16] R. J. van Glabbeek, The linear time — branching time spectrum ii, in: E. Best (Ed.), CONCUR'93, Springer Berlin Heidelberg, Berlin, Heidelberg, 1993, pp. 66–81.

[17] R. J. van Glabbeek, The linear time - branching time spectrum, in: J. C. M. Baeten, J. W. Klop (Eds.), CONCUR '90 Theories of Concurrency: Unification and Extension, Springer Berlin Heidelberg, Berlin, Heidelberg, 1990, pp. 278–297.

[18] M. Abadi, L. Cardelli, An imperative object calculus, in: P. D. Mosses, M. Nielsen, M. I. Schwartzbach (Eds.), TAPSOFT '95: Theory and Practice of Software Development, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 469–485.

[19] M. Abadi, L. Cardelli, A Theory of Objects, Monographs in Computer Science, Springer-Verlag New York, 1996.

[20] K. Peters, U. Nestmann, U. Goltz, On distributability in process calculi, in: M. Felleisen, P. Gardner (Eds.), Programming Languages and Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 310–329.

[21] D. Gorla, Towards a unified approach to encodability and separation results for process calculi, Inf. Comput. 208 (9) (2010) 1031–1053. doi:10.1016/j.ic.2010.05.002.

[22] D. Gorla, U. Nestmann, Full abstraction for expressiveness: history, myths and facts, Math. Struct. Comput. Sci. 26 (4) (2016) 639–654. doi:10.1017/S0960129514000279.

[23] B. Bisping, U. Nestmann, Computing coupled similarity, in: T. Vojnar, L. Zhang (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer International Publishing, Cham, 2019, pp. 244–261.

[24] K. Peters, R. J. van Glabbeek, Analysing and comparing encodability criteria, in: S. Crafa, D. Gebler (Eds.), Proceedings of the Combined 22th International Workshop on Expressiveness in Concurrency and 12th Workshop on Structural Operational Semantics, EXPRESS/SOS 2015, Madrid, Spain, 31st August 2015, Vol. 190 of EPTCS, 2015, pp. 46–60. doi:10.4204/EPTCS.190.4.

[25] B. Bisping, U. Nestmann, K. Peters, Coupled similarity: the first 32 years, Acta Informatica 57 (3-5) (2020) 439–463. doi:10.1007/s00236-019-00356-4.

[26] M. Merro, J. Kleist, U. Nestmann, Local π-calculus at work: Mobile objects as mobile processes, in: J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, T. Ito (Eds.), Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 390–408.

[27] M. Merro, D. Sangiorgi, On asynchrony in name-passing calculi, in: K. G. Larsen, S. Skyum, G. Winskel (Eds.), Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 856–867.

[28] C. Fournet, G. Gonthier, The reflexive cham and the join-calculus, in: Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '96, Association for Computing Machinery, New York, NY, USA, 1996, p. 372–385. doi:10.1145/237721.237805.

[29] C. Hewitt, P. B. Bishop, R. Steiger, A universal modular ACTOR formalism for artificial intelligence, in: N. J. Nilsson (Ed.), Proceedings of the 3rd

International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973, William Kaufmann, 1973, pp. 235–245. URL `http://ijcai.org/Proceedings/73/Papers/027B.pdf`

[30] G. Agha, P. Thati, An algebraic theory of actors and its application to a simple object-based language, in: O. Owe, S. Krogdahl, T. Lyche (Eds.), From Object-Orientation to Formal Methods: Essays in Memory of Ole-Johan Dahl, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 26–57.

[31] R. van Glabbeek, U. Goltz, J.-W. Schicke, On synchronous and asynchronous interaction in distributed systems, in: E. Ochmański, J. Tyszkiewicz (Eds.), Mathematical Foundations of Computer Science 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 16–35.

[32] Proceedings combined 27th international workshop on expressiveness in concurrency and 17th workshop on structural operational semantics, Electronic Proceedings in Theoretical Computer Science 322 (Aug 2020). doi:10.4204/eptcs.322.

[33] R. Griesemer, R. Pike, K. Thompson, I. Taylor, R. Cox, J. Kim, A. Langley, Hey! ho! let's go!, `https://opensource.googleblog.com/2009/11/hey-ho-lets-go.html` (2009).

[34] The Go Team, The Go Programming Language Specification, `https://golang.org/ref/spec` (2021).

[35] G. N. Buckley, A. Silberschatz, An effective implementation for the generalized input-output construct of CSP, ACM Trans. Program. Lang. Syst. 5 (2) (1983) 223–235. doi:10.1145/69624.357208.

[36] D. Kumar, A. Silberschatz, A counter-example to an algorithm for the generalized input-output construct of CSP, Inf. Process. Lett. 61 (6) (1997) 287. doi:10.1016/S0020-0190(97)00040-9.

[37] K. Claessen, J. Hughes, Quickcheck: A lightweight tool for random testing of haskell programs, in: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming, ICFP '00, Association for Computing Machinery, New York, NY, USA, 2000, p. 268–279. doi:10.1145/351240.351266.