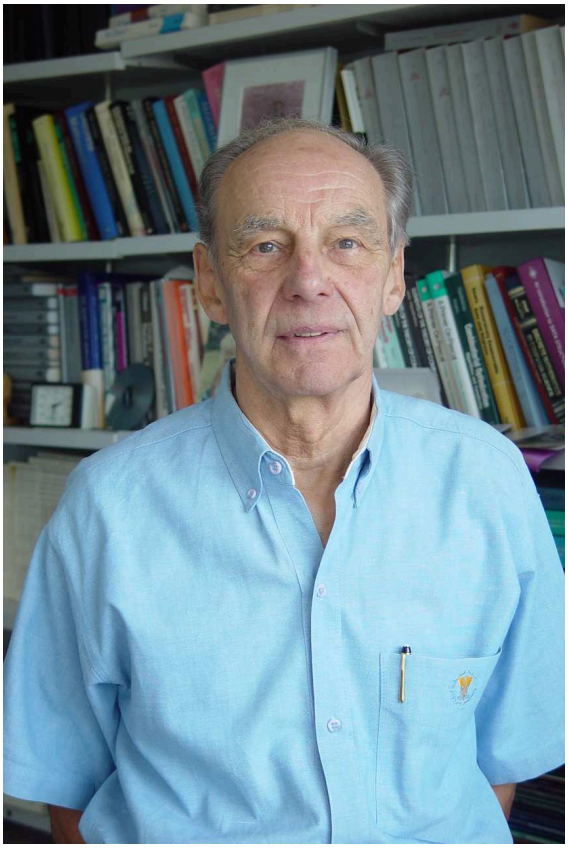
SIGACT News Complexity Theory Column 115:

Juris Hartmanis and Two Golden Rules

Lane A. Hemaspaandra*
Dept. of Computer Science, University of Rochester, Rochester, NY 14627, USA



Juris Hartmanis was so preternaturally wise and brilliant that in time people may find it hard to believe that someone so extraordinary existed in this world. Yet as I write this, three months to the day after Juris passed away, it still seems impossible that such a force of nature can no longer be part of this world.

Juris's work will live on forever. But even his youngest PhD advisees are nearer the end of their careers than the beginning. We each surely do our best to pass down to our own students what we learned from Juris about how to do research in computer science, and with luck our students will similarly pass that on. This article's goal is to write down and pass on what I feel are two of the most important aspects of what Juris conveyed as to how computer science research should be done and presented.

Before turning to that, let me briefly mention what is coming in future issues of this column. The next issue of *SIGACT News* will have two articles in its complexity column: an article by Eric Allender, titled "Parting Thoughts and Parting Shots (Read On for Details on How to Win Valuable Prizes!)," and a thank-you-all article by me, as that issue will be my final one as editor of the complexity column. It has been

*Supported in part by grant NSF-CCF-2006496.

one of the great treats of my career to have served as the complexity column's editor for these thirty years, and to have been able to interact with so many wonderful guest-article writers. The column's future is tremendously bright, since *starting with the June 2023 issue the column's editor will be the terrific Ben Lee Volk*. Ben Lee has already lined up exciting articles for many issues to come, including (these titles/topics are tentative):

- Lijie Chen and Roei Tell on "New Directions in Derandomization,"
- Sumegha Garg on "Memory-Sample Tradeoffs for Learning," and
- Sevag Gharibian on "The Seven Faces of Quantum NP."

As you know, dear reader, this issue of *SIGACT News* is a special one in honor of Juris Hartmanis. The previous issue's complexity column included a tribute ("Juris Hartmanis (1928–2022): Understanding Time, Space, and Human Creativity") that I wrote on Juris. Although in this issue's article there is some content overlap with that tribute, the goal of this issue's column is different, and in particular is to point to two important takeaways from Juris's career and words, as to how computer science should be done. Although Juris was one-of-a-kind, his approach to doing research and writing up research is a gift from him from which we all can benefit.

This article will use the term "golden rule" not in its ethical-rule sense but in its "a fundamental principle to be followed in order to ensure success in general or in a particular undertaking" sense [AH18], although both golden rules covered in fact are less about seeking "success" and more about how computer science research *should* be done and communicated (though the two golden rules in fact do support the success of the field and of those individuals who take them to heart).[1]

The first golden rule telegraphed by Juris's career is that **computer science research needs to be built on rigorous, clear foundations.** He wrote in his 1981 *Annals of the History of Computing* article [Har81, p. 50] "In computer science we must first imagine or build what we want to study and then develop the concepts, formalizations, and theories to be able to think about the possible systems and solutions and pick a good one from them. We must develop the intellectual tools not only to explore the existing but also to study the possible—to help us imagine it, analyze it, and build it." However, importantly, he did not mean that to apply just to research in theoretical computer science (despite the fact that that article was about the development of theoretical computer science). He felt that all areas of computer science (and even adjacent areas) are best studied based on developing real, deep, crisp foundations, frameworks, and approaches. For example, speaking about I-schools in his 2009 interview with William Aspray [AH10], Juris said, "So let me say what I certainly believe they shouldn't be. They shouldn't be computer science light—you know, this should not be watered-down computer science. They should have the same good deep intellectual content as computer science has created in most of its areas."

Some people feel that very foundationally/mathematically grounded research in some non-theory area X of computer science is best viewed as *theoretical computer science research*. But my

[1] Though both golden rules (or certainly the first, and also the second at least in its second form) that I am extracting are clearly grounded either in Juris's writings or things he often said, I should mention that I am reading Juris's view on the applicability of the first rule broadly (though my sense is that that is exactly what Juris thought: that all of the many subareas of computer science and its related fields ideally should have crisp, formalized, appropriate foundations).

impression was always that Juris felt that such work in fact was *(well-grounded) research in X*, and that all areas X within computer science ideally should build well-grounded foundations.[2]

None of that is to say that Juris didn't have a very warm spot for giving core theory training to all undergraduate computer science students. Speaking in that same Aspray interview of a change at Cornell regarding the undergraduate computer science major and whether it required a "models of computation" course, he said, "That course is not required anymore for our undergraduate students. It was required until a year ago or so. Now, I do agree that courses must change, and it has changed as time goes on. But I am sorry that it's gone [as a requirement of the major]. It will be taught [as an elective] and many students will still take it, but I thought of the Cornell tradition in theory—it's good for your soul, it makes you think more clearly."

Juris's belief in clarity, theory, and foundations—that ideally all areas of computer science should be grounded in clear, and generally mathematical or at least formal foundations, and that claims or even conjectures should be crisply, formally expressed, so that claimed proofs of them can be sought and evaluated—did not however mean that Juris always disliked the case of where one obtained answers/solutions in a way that didn't come with explanations. Rather, in such a case he would want one to prove or gain insight into whether one could provide explanations, or into why doing so was unlikely (e.g., doing so would imply some shockingly unlikely complexity-class collapses).

In fact, one result he loved teaching—and taught to countless graduate students from all areas of computer science—was about a case where one might, counterintuitively, have answers yet not have certificates/explanations of exactly how the answers were correct! However, it was a great example of a case where one at a higher level had an explanation for why one could not find explanations. I refer to the amazing Borodin–Demers Theorem (see [BD76], and for a discussion of the result's provenance, background, and history, and of related work, see also footnote 9 of [HHM20]).

Briefly put (a more formal statement will be given in an included image later in this article), the Borodin–Demers Theorem states that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ (a hypothesis we all believe is true, since if not then integer factoring is in P, and thus much of cryptography falls), implies that there is a set S of obviously satisfiable formulas such that no polynomial-time machine can find their satisfying assignments. (So this is a set of formulas all of whose satisfiability is obvious, yet such that, relative to any (deterministic) polynomial-time satisfying-assignment seeking program, for infinitely many of the formulas it is not obvious *how* they are satisfied.) This is exciting stuff, since the theorem is of the form "A implies B," where A is something most people feel is true but B is something that many of the same people would, before knowing of the theorem, suspect to be false.

There is even a twist within that twist, since many bright students, hearing the theorem, would

[2]The view that all areas of computer science should be approached foundationally and rigorously was not just Juris's, but was (probably in large part due to Juris's guiding influence in the shaping of the department) the shared worldview of Cornell's computer science department back when I was a student there. For example, the other two computer science professors on my PhD advisory committee were John Hopcroft, who at the time was formally exploring the complexity of key planning challenges in robotics [JJW84, HJW85], and David Gries, whose many passions included helping students approach computer programming as science rather than as an art [Gri81] (and, from different directions, other computer science faculty of the time—e.g., Professors Constable and Demers—also focused on foundationally exploring the nature of programs and programming.

say, "But search and decision are well known to be polynomial-time Turing-interreducible for SAT, due to SAT's 2-disjunctive self-reducibility. So the right-hand side of the Borodin–Demers Theorem simply can't hold." And that reasoning is very close to being correct. The right-hand side of the Borodin–Demers Theorem indeed must defy self-reducibility if it wishes to hold. It does! The twist here is that the set S may have formulas such that some of their children created by variable setting and simplification *are not in* S; thus the self-reducibility approach won't work, because there is no promise that all paths of the self-reducibility tree are addressed by S.

From Juris's joy at teaching that result, and from his writings and lectures, one can't miss the second golden rule: **Convey the beauty and drama of results.** Or, somewhat differently put, and as he often advised students, **A paper should tell a good story!**

This might seem obvious and easy to achieve. But although it perhaps is obvious, it nonetheless is extremely hard to achieve. Yet Juris framed, found, and—with charisma, creativity, and outright joy—told amazing stories in his research and his teaching.[3] And Juris's example and advice inspired those of us who knew him to do our admittedly very limited best to try to follow the two golden rules I've mentioned in this article. Simply put and as mentioned earlier in this article, Juris's sharing of his worldview with the next generations was a tremendous gift to the field and to individuals.

Juris of course did not publish a proof of the Borodin–Demers Theorem (rather, it was a beautiful result that he loved teaching, and that he presented a proof of as he taught). Myself, I was so taken by Juris's teaching of the Borodin–Demers Theorem that, now as a professor myself, I usually teach it as the very end my school's undergraduate models of computation course (namely, at the end of the unit introducing NP and complexity—it in fact makes up the final two slides of that unit), and I also cover it near the start of our graduate complexity theory course. In making slides to cover it, I tried to remain as true as possible to my memory of how Juris conveyed it (in class, with him writing on a blackboard, and of course without him drawing pointed teeth or any kibitzing characters). The theorem really is rather amazing: It is a rather profound claim, yet its proof fits on one slide. (The only thing one has to also know is that, as is introduced a few slides earlier in the courses, $\widehat{F}$ represents a version of the Cook–Karp–Levin reduction whose output formulas openly reveal from what machine and input the formula was mapped to (see [Gal74]).[4]) Here are those two slides—the first stating the theorem and the second containing the proof.[5]

---

[3]The same holds for the first golden rule: Some, who share that rule's view, might feel that the need for rigorous foundations and formalizations is obvious. Yet it in fact is extremely hard to insightfully build foundations and formalisms that get things just right. But, nonetheless, Juris was so talented in sculpting the right foundations and formalisms that he did so unerringly and seemingly effortlessly.

[4]In more detail, let N_i be fixed, nice enumeration of NPTMs, covering all NP languages, that has the property that the running time of each N_i is bounded by $|x|^i + i$. The map from NPTMs N_i and inputs x to $\widehat{F}_{i,x}$ that we are speaking of is one that satisfies (a) $N_i(x)$ accepts $\iff$ $\widehat{F}_{i,x}$ is satisfiable, (b) we can produce $\widehat{F}_{i,x}$ in time polynomial in $|N_i| + |x|^i + i$, and (c) from $\widehat{F}_{i,x}$ we can in polynomial time recover N_i and x. (The two polynomial algorithms referred to there are uniformly polynomial algorithms applying over all i and x—not different ones with different degrees for each different i.)

[5]One may naturally wonder whether the converse of the Borodin–Demers Theorem holds. That remains open to this day. However, Impagliazzo and Naor [IN88] showed that there are relativized worlds in which a version of the converse fails to hold, and the PhD thesis I wrote under Juris's supervision proved that for the UP analogue of the Borodin–Demers Theorem both the theorem and its converse hold, i.e., we have a complete characterization [Hem87, Chapter 6.2].

Thm. (the Borodin-Demers Theorem) $NP \cap coNP \neq P$

$\Downarrow$

$(\exists A)[A \in P \wedge A \subseteq SAT \wedge$ "no poly-time machine can find solutions to all of A" $]$.

i.e. $(\forall$ poly-time comp. fcn $f)$
$(\exists q \in A)[\, f(q)$ is not a satisfying assignment of $q \,]$.

Note: $(\exists A)[\, \|A\| = \infty \wedge A \in P \wedge A \subseteq SAT \,]$ hold unconditionally, e.g., $A = \{T, TvT, TvTvT, \ldots\}$.
It is just due to also having the 3rd Conjunct that this theorem has (very) sharp teeth.

Proof of the Borodin-Demers Theorem:
We assumed $NP \cap coNP \neq P$. So, let $L \in (NP \cap coNP) - P$.
Let N_k be an NPTM (from the std. enum.) accepting L.
Let $N_{\hat{k}}$ " " " " " " accepting $\overline{L}$.

Set: $S = \{ f \mid (\exists x)[\, f \text{ is } (\hat{F}_{k,x}) \vee (\hat{F}_{\hat{k},x}) \,] \}$.
This completes the construction!!—it took just a few lines.!!!

Claim $S \in P$.

Claim $S \subseteq SAT$.
Pf. $(\forall x)[\, x \in L \vee x \notin L \,]$.
So, $(\forall x)[\, (\hat{F}_{k,x}) \vee (\hat{F}_{\hat{k},x})$ is satisfiable $]$.

Claim If you could efficiently (Ptime) find satisfying assignments for S, then you could determine membership in L in poly time. (Why?)
(But that yields a contradiction, as $L \notin P$ was assumed.)
i.e., α

$Q \in \triangleright$

In conclusion, I'd urge all of us—including you, dear reader—to keep in mind and pass on Juris Hartmanis's wise view of computer science, so that not just we but also the students we

teach view foundations as central throughout computer science, and so that we and our students do our best to have each of our papers tell a good story.

References

[AH10] W. Aspray and J. Hartmanis. "Dr. Juris Hartmanis interview: July 26, 2009; Cornell University in Ithaca, New York". In *ACM Oral History Interviews*, Interview No. 19, 77 pages, April 2010. Association for Computing Machinery.

[AH18] *The American Heritage Dictionary of the English Language*. Collins Reference, 5th edition, 2018.

[BD76] A. Borodin and A. Demers. Some comments on functional self-reducibility and the NP hierarchy. Technical Report TR 76-284, Department of Computer Science, Cornell University, Ithaca, NY, July 1976.

[Gal74] Z. Galil. On some direct encodings of nondeterministic Turing machines operating in polynomial time into P-complete problems. *SIGACT News*, 6(1):19–24, 1974.

[Gri81] D. Gries. *The Science of Programming*. Texts and Monographs in Computer Science. Springer, 1981.

[Har81] J. Hartmanis. Observations about the development of theoretical computer science. *Annals of the History of Computing*, 3(1):42–51, 1981.

[Hem87] L. Hemachandra. *Counting in Structural Complexity Theory*. PhD thesis, Cornell University, Ithaca, NY, May 1987. Available as Cornell Department of Computer Science Technical Report TR87-840.

[HHM20] E. Hemaspaandra, L. Hemaspaandra, and C. Menton. Search versus decision for election manipulation problems. *ACM Transactions on Computation Theory*, 12(#1, Article 3):1–42, 2020.

[HJW85] J. Hopcroft, D. Joseph, and S. Whitesides. On the movement of robot arms in 2-dimensional bounded regions. *SIAM Journal on Computing*, 14(2):315–333, 1985.

[IN88] R. Impagliazzo and M. Naor. Decision trees and downward closures. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, pages 29–38. IEEE Computer Society Press, June 1988.

[JJW84] J.Hopcroft, D. Joseph, and S. Whitesides. Movement problems for 2-dimensional linkages. *SIAM Journal on Computing*, 13(3):610–629, 1984.