How Important are Formal Methods and Formal Logic for Software Engineering Education?

Antti Valmari & Veikko Halttunen
University of Jyväskylä, Faculty of Information Technology

1 Introduction

Starting point: a discrepancy

- ▷ many consider logic as underlying software engineering and scientific thinking
  - logic is standard material in computer science / SWE degrees
- ▷ formal methods address major problems related to SW quality
  - getting the specification right, and implementing it correctly
- ◁ formal logic and formal methods are not used much in practice

We discuss

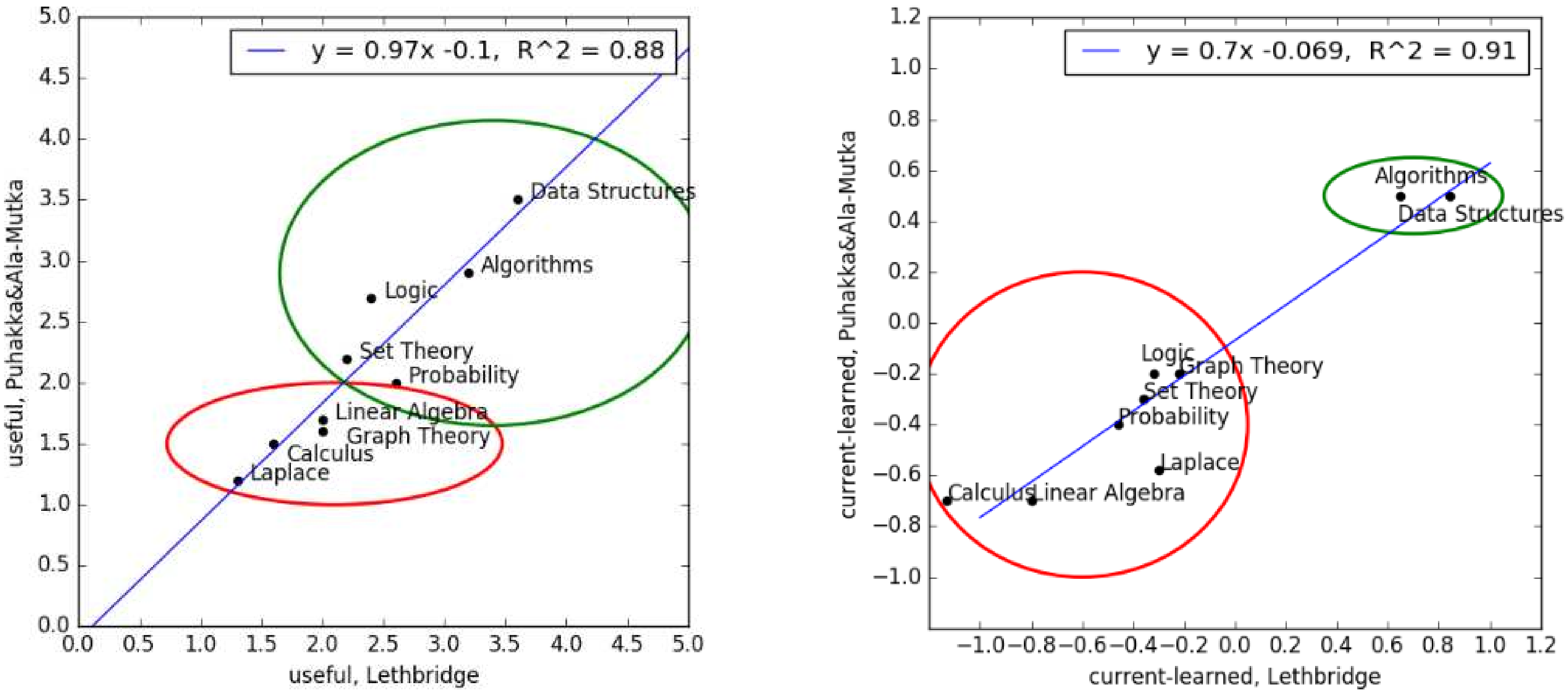
- how professionals perceive the importance of logic and formal methods
- how much they are taught
- how well they work in practice
- *why they do not work better than that*

first-order logic	second-order logic
less expressive: $\forall x$, $\exists x$	more expressive: $\forall$, $\exists$ $P(\ldots)$, $f(\ldots)$
easier to reason with, complete proof systems	more difficult to reason with

2 What Surveys Say

There are (apparently only) five surveys on perceived math needs in SW

- very different
- 2000, 2005, 2004/2009, 2007, 2020
- each suffers from weaknesses in sample size, geographical representability, etc.
- the messages in all of them regarding math, logic and formal methods are very similar



Niemelä, P. & Valmari, A.: Elementary Math to Close the Digital Skills Gap, CSEDU 2018

3 What Curricula Recommendations Say

IEEE / ACM Software Engineering 2014

- 467 “lecture hours” of “what every SE graduate must know”
  $\rightarrow$ of them 50 “Mathematical foundations”
  $\rightarrow$ within which “Basic logic (propositional and predicate)”
- “desirable” “essential”
- “knowledge” “comprehension” “application”
- “logic and discrete mathematics should be taught in the context of their application”
- formal methods are mentioned, but given little emphasis
  - cf. testing 18 hours

ACM / IEEE / AAAI Computer Science Curricula 2023

- recent enough to reflect data science and quantum computing (and generative AI?)
- significant background surveys
  - 865 industry + 427 educator respondents on a wide range of topics [2021]
  - 597 educator respondents on math [2022]
- "lecture hours"

"lecture hours"	obligatory	should-be-but-cannot-be obligatory	
altogether	270	483	
math & statistics	55	145	cf. Computer Science 2013: 37 + 4
discrete math	29	11	includes logic
probability	11	29	
statistics	10	30	
linear algebra	5	35	
calculus	0	40	

  - "application of mathematics has increased"
  - however, "mathematics should not be the reason why otherwise well-qualified students are kept away from computer science"
- only propositional and "simple predicate logic" are covered
- informal (= ordinary math) proof techniques
- formal methods are "Non-Core"

4 Bottleneck: Writing Convincing Formal Specifications

Formally specifying sorting is trivial — or is it?

- the following

$$\forall i\,;\, 1 \le i < n : A[i-1] \le A[i]$$

  does not rule out $\quad$ **for** $i := 1$ **to** $n-1$ **do** $A[i] := A[0]$

- the following

$$(\forall i\,;\, 0 \le i < n : \exists j\,;\, 0 \le j < n : B[j] = A[i]) \wedge$$
$$(\forall i\,;\, 0 \le i < n : \exists j\,;\, 0 \le j < n : A[j] = B[i])$$

  allows outputting $[1,2,2]$ given $[1,1,2]$

- the following

$$\exists f : \forall i\,;\, 0 \le i < n : 0 \le f(i) < n \wedge B[i] = A[f(i)] \wedge \exists j\,;\, 0 \le j < n : i = f(j)$$

  requires second-order logic, and how to become convinced that it is correct?

- the following

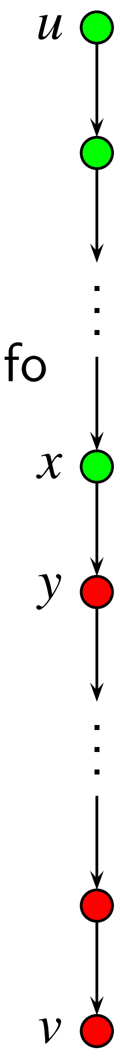
$$\forall x : \mathrm{number\_of}(x, A) = \mathrm{number\_of}(x, B)$$

  requires both array element type and $\mathbb{N}$, and special (application-specific?) notation

- and we have not even started discussing *stable* sorting

Reachability

- central in graph algorithms, memory management, ...
- theorem: cannot be specified in first-order logic without some strong help
- second-order: $\forall P : \neg P(u) \vee P(v) \vee \exists x : \exists y : P(x) \wedge (x \rightsquigarrow y) \wedge \neg P(y)$
  - how to become convinced that it is correct?



Fairness

- e.g., every submitted paper must eventually be reviewed, but not necessarily fifo
- amazingly difficult to specify
  - e.g., how to rule out solutions that prevent from submitting?

Observations

- it is often difficult or impossible to find a straightforward formalization

⇒ it is often difficult to see whether what a formal spec says is right

⇒ informal spec & informal proof may be much more convincing than formal spec and automated proof

- how to know that a spec, formal or informal, says everything essential?

5 Bottleneck and Strength: Automatic Verification

"Nearly all binary searches and mergesorts are broken" [Bloch 2006]

- arithmetic overflow when computing `int mid = (low + high) / 2;`
- occurs only with very big arrays
  $\Rightarrow$ remained undetected for 9 years or so, until computer memories grew big enough

Strength

- checks numerous routine details more reliably than humans
- as a by-product, may confirm the correctness of the abstract algorithm
- may help in validating requirements (verify ad-hoc desired properties)

Bottlenecks: (1) formalization of the spec (2) significant amount of human work needed

- big lines in the proof
- occasional details: 2 228 / 372 307 in [de Gouw & al. 2014] counting & radix sort
  `res[c[a[j]]] = a[j];` $\rightsquigarrow$ `int tmp = a[j]; res[c[tmp]] = tmp;`

[Beckert & al. 2024] highly optimized sorting algoritm, $>$ 900 lines of Java

- the specification and guiding the proof: 2 500 lines of JML
- 4 person-months

6 Undefined Expressions

Underspecification [Gries & Schneider 1995] is widely used in two-valued logic

- every expression always has a value in the domain, but we do not always know it
- does not tell if 0 is a root of $\frac{1}{x} = 3$
- makes 0 a root of $\frac{1}{x} = \frac{x}{2} + \frac{1}{2x}$

Short-circuit "and" and "or"

- very common: `&&` and `||`
- not commutative, unlike $\wedge$ and $\vee$
- precise match in three-valued logic: $P \wedge (\neg P \vee Q)$ and $P \vee (\neg P \wedge Q)$

Also some other things become much more natural in three-valued logic

[Chalin 2005]

- $>$ 200 software professional respondents

when `a[0]` does not exist	true	false	error / except.	other
`a[0] == 0 || a[0] != 0`	8 %	10 %	74 %	7 %
`a[0] == a[0]`	16 %	7 %	75 %	3 %

- "two-valued logic is misaligned with programming practice"

7 Concluding Remarks

Mathematical thinking ⇝ lightweight formal ⇝ fully formal

At the propositional logic level, focus on common sense and common misunderstandings

- mainstream math, theoretical CS and programming do not use truth tables, etc.
  ⇒ do not waste time on them
- prone to misunderstandings:
  - principle of explosion and its variants
  - "if . . . then . . ." is unidirectional
  - "if . . . then . . ." is often better treated as a reasoning rule, not as $\neg P \vee Q$
  - logical equivalence (← ↑ these two might be worth a paper of its own)

~~I am a programmer~~
~~All progr.s make programming errors~~
~~I make programming errors~~

Tools that make it easier to specify formally ⇒ worth teaching, if you favour formality

- three-valued logic
- second-order logic

Teaching formal proof systems is reasonable only if aiming at full formality

A wonderful tool for teaching logic has been presented in this workshop!

Thank You for attention! **Questions, discussion?**