Using Domain-Independent Problems for Introducing Formal Methods

Raymond Boute

INTEC, Universiteit Gent, Belgium,
Raymond.Boute@intec.UGent.be

Abstract. The key to the integration of formal methods into engineering practice is education. In teaching, domain-independent problems — i.e., not requiring prior engineering background— offer many advantages.

Such problems are widely available, but this paper adds two dimensions that are lacking in typical solutions yet are crucial to formal methods: (i) the translation of informal statements into formal expressions; (ii) the role of formal calculation (including proofs) in exposing risks or misunderstandings and in discovering pathways to solutions.

A few example problems illustrate this: (a) a small logical one showing the importance of fully capturing informal statements; (b) a combinatorial one showing how, in going from "real-world" formulations to mathematical ones, formal methods can cover more aspects than classical mathematics, and a half-page formal program semantics suitable for beginners is presented as a support; (c) a larger one showing how a single problem can contain enough elements to serve as a Leitmotiv for all notational and reasoning issues in a complete introductory course.

An important final observation is that, in teaching formal methods, no approach can be a substitute for an open mind, as extreme mathphobia appears resistant to any motivation.

Index terms — Domain-independent problems, Formal methods, Functional Predicate Calculus, Funmath, Generic functionals, Teaching, Specification, Word problems

1 Introduction: motivation and overview

A gap in engineering professionalism One often hears the complaint that the use of formal methods into everyday software engineering practice is taking a long time in becoming commonplace (except for critical and some embedded systems) and that, as a result, professionalism in software design is generally low.

Yet, the published literature reports many projects for which formal methods were essential or at least the key to success. Why, then, is the software industry at large so slow in exploiting the advantages?

Many explanations have been conjectured by diverse people, but the following one seems inescapable as "Occam's razor": universities are providing far from sufficient education in formal methods to generate the massive injection of qualified people necessary for enabling industry to integrate formal methods into

their everyday software design practice. The success stories remain restricted to companies that are either consultants specialized in the area of formal methods or users of ad hoc support for specific projects from consultants and universities.

The contrast with classical engineering disciplines, in particular electrical engineering, is significant. Mathematical modelling has not only proven indispensable for todays communications technology, but has been accepted de facto since centuries as an essential part of engineering education. It is commonplace in industry, and no university engineer would dare confessing to his project leader that he doesn't know enough elementary calculus or algebra to cope with this.

Yet, some universities still turn out software "engineers" whose grasp of logic is not better than high school level, and whose highest abstraction "tool" for specifying systems is some program-like code which they are unable to model and analyze mathematically. Complaints from industry are not surprising [19]. A serious educational obstacle pointed out by one of the reviewers is that logic is much harder than differential calculus, as also indicated by other studies [1].

On using tools A similar gap exists in the ability to use software tools judiciously. High school education provides sufficient mathematics for starting to use tools for classical mathematics like Mathematica, Maple, Matlab, Mathcad with ease, and a good calculus or analysis course at the freshman and junior level prepares for more advanced use by providing a solid basis and insight (not a luxury, given the many "bugs" reported in the literature). Mathematicians and engineers 150 years ago would have no difficulty in using today's tools without a "tutorial".

By contrast, for software tools supporting CS-oriented formal methods, classical mathematics offers no preparation, and too many computing curricula do not even start filling this gap. Using the tools themselves in an introductory course as a vehicle for introducing (or, worse, as a surrogate for) the relevant mathematics is highly inappropriate[1] since such tools are still very design-specific[2] and hence induce a narrow and misleading view in beginners, turning them into sorcerer's apprentices. Of course, tools can be very useful as an illustration, especially for the shortcomings in and the differences between them. In fact, in the same vein some of the best analysis texts provide exercises with tools precisely to show the pitfalls [21], and always keep the mathematics and "thinking" central.

In brief: the best preparation for using tools is solid mathematics education.

Curriculum design Providing the relevant mathematics early creates the opportunity for other computer-oriented courses (HW and SW) to start using serious mathematical modeling, rather than remaining stuck at the old descriptive level with some elementary programming exercises as the highest intellectual activity. For classical mathematics, preparation usually starts in high school but, as universities have no control at this level, the earliest opportunity to teach the basic

[1] This observation assumes today's state of the art; only vast progress can alter it.

[2] Unlike software tools for classical mathematics, which support mature notational and formal calculation conventions, current tools in the formal methods area are still based on overly specific logics and too reflective of various implementation decisions.

mathematics for computing science and engineering is the freshman level. A typical embedding in a traditional curriculum is achieved by thoroughly changing the content of an existing discrete mathematics course for this purpose, as exemplified in the textbook by Gries and Schneider [11] and the BESEME (BEtter Software Engineering through Mathematics Education) project [15].

To summarize, an early start paves the way for "weaving formal methods into the undergraduate curriculum" as advocated by Wing [23]. The extent to which other courses catch this opportunity depends on the quality of the staff, and determines the ability of the students to use formal methods.

The role of domain-independent problems Providing the mathematical basis for CS early in the curriculum is facilitated by assuming no prior technical or engineering background, as is also customary in calculus courses. An independent reason is the principle of separation of concerns: not overburdening the student by combining new mathematical with new (or recent) engineering concepts.

Any basic course needs illustrations and, more importantly, problems to enhance insight in the concepts and to learn using them in problem solving.

Domain-independent problems combine all the above requirements. They can be understood by anyone, and are fun to solve. Furthermore, they are widely available in both educational [2, 11] and recreational literature [9, 17]. Finally, courses and textbooks on this basis can reach more easily over boundaries between disciplines. One might ask how this pertains to CS-oriented mathematics, given its specialistic reputation. Actually, this reputation is undeserved, given the rapidly growing evidence that the insights and reasoning styles fostered by CS have wide applicability in mathematics [8], in science and in engineering [5].

New dimensions in solving domain-independent problems Concerns arising from the application of formal methods add new dimensions to problem solving that are also best illustrated by domain-independent problems.

Indeed, solutions to such problems, especially "puzzles", are traditionally often presented with a "look how clever" undertone of impressing the audience. Unfortunately, this effect often comes at the expense of hiding steps in the calculation or in the conversion from the informal statement to mathematical formulas. Sometimes this is forgivable, e.g., when common notation falls short.

When using formal methods in practice, avoiding or exposing hidden steps and assumptions is part of the task. Hence, in the introduction of formal methods, domain-independent problems can help emphasizing the following two important issues: (i) the translation of informal statements into formal expressions; (ii) the role of formal calculation (including proofs) in exposing misunderstandings or possible risks and in discovering pathways to solutions.

Scope of this paper and approach We will show that even small domain-independent problems can have sufficiently rich ramifications for illustrating central issues in formal methods, and that medium ones can contain enough elements to serve as the Leitmotiv for illustrating all notational and reasoning issues in a complete introductory course on basic mathematics for CS.

For notation and reasoning we use *Funmath* (Functional Mathematics) [3, 5], a very general formalism unifying mathematics for classical engineering and CS.

The language [3] uses just four constructs, yet suffices to synthesize familiar notations (minus the defects) as well as new ones. It supports formal calculation rules convenient for hand calculation and amenable to automation.

The reasoning framework has two main elements. First, concrete generic functionals [4] support smooth transition between pointwise and point-free formulations, facilitating calculation with functionals and exploiting formal commonalities between various engineering mathematics. Second, a functional predicate calculus [5] makes formal logic practical for engineers, allowing them to calculate with predicates and quantifiers as fluently as with derivatives and integrals.

Here we use the language mostly in its "conservative mode", restricted to expressions with the same look and feel as common mathematical conventions. We only lift this restriction when common notation cannot express what is needed.

As a result, most of this paper requires neither prior knowledge of nor introduction to Funmath, and we can refer to [5] for details or for exploring deeper.

Overview We consider some selected issues only; any attempt at completeness would rapidly grow into a textbook. Section 2 uses a very small word problem (at the level of propositional logic) to highlight some psychological and educational issues and to demonstrate the importance of completeness when capturing informal statements. Section 3 shows via a small combinatorial problem how steps in the transition from "real-world" formulations to mathematical ones are missed in traditional solutions, yet can be captured by formal methods. A half-page formal program semantics suitable for beginners is presented as a support. Section 4 shows how a single problem can give rise to enough topics for a complete introductory course on formal methods. Section 5 presents some conclusions and observes that, in teaching formal methods, no approach can be a substitute for an open mind, as extreme mathphobia appears resistant to any motivation.

2 On logic and properly formalizing informal statements

Most people would attribute to themselves a basic "natural" understanding of logic. Yet, studies by Johnson-Laird [13] about logic reasoning by humans expose serious flaws. Table 1 describes two typical experiments and their outcomes.

One step further: most engineers and mathematicians would consider themselves fairly fluent in logic by profession[3]. We have no data on how this group as a whole would perform on the test, but experience with the subgroup in computer science/engineering —where logic is vital— gives reason for concern.

Indeed, we found that even CS students who previously had a full semester course on formal logic elsewhere generally did poorly on this test.

[3] Introductions to logic that are too elementary (as in traditional discrete math courses) only strengthen this feeling, since they offer little more than a semi-formal notation or even syncopation [20] for expressing something they were already doing informally in mathematics before and will continue doing informally afterwards.

Table 1. Two experiments as reported by Johnson-Laird

(a)	One of the following assertions is true about a particular hand of cards, and one of them is false about the same hand of cards: If there is a king in the hand, then there is an ace in the hand If there isn't a king in the hand, then there is an ace in the hand. Q: What follows? Subjects overwhelmingly infer that there is an ace in the hand.
(b)	Only one of the following assertions is true about a particular hand of cards: There is a king in the hand, or an ace, or both. There is a queen in the hand, or an ace, or both. There is a jack in the hand, or a ten, or both. Q: Is it possible that there is an ace in the hand? Nearly every participant in our experiment responded: 'yes'.

Analysis of the answers suggests that here some other effect is responsible than the one Johnson-Laird observes in people without logic background. Indeed, the most common error for problem (a) was taking the conjunction of the two assertions listed. The errors for problem (b) were more diverse. However, the answers indicated that often the "preamble" (the part of the problem statement before the list of assertions) was simply ignored. Even students who attempted formalizing the problem statement left the preamble out of this process.

In general there seems to be a strong tendency to skip parts of the problem statement (which are perhaps perceived as mere padding) and, as a result, "jumping to conclusions". It is safe assuming that the same effect occurs with more complex specifications stated as many pages of text. Recently, Vaandrager mentioned that IEEE specifications of complex protocols are typically stated in words, with at best an appendix where fragments are formalized [22].

We suggest the following discipline to eliminate, or at least reduce, this effect: start by formalizing every sentence separately as accurately as the formalism used permits, and simplify or combine only afterwards. . In particular, discard seemingly irrelevant parts only if due consideration justifies doing so.

For instance, in solving (a), do not directly formalize the problem statement in one step as $(k \Rightarrow a) \oplus (\neg\, k \Rightarrow a) \equiv 1$ (the identifier conventions are obvious). Instead, in a first version, maintain one-to-one correspondence with the text, as illustrated in the following set of equations, and simplify afterwards.

$$\begin{aligned} \alpha \oplus \beta &\equiv 1 \\ \alpha &\equiv k \Rightarrow a \\ \beta &\equiv \neg\, k \Rightarrow a \end{aligned}$$

As an aside: in programming, one discourages writing `if b = true then ...` since `if b then ...` is better style. In logic, it is often better style to give equations the shape of equations; so we wrote $\alpha \oplus \beta \equiv 1$ rather than just $\alpha \oplus \beta$.

More importantly, in view of faithfully formalizing informal statements, one might argue that using $\oplus$ in the first equation already skips ahead of things,

since the preamble is a conjunction of two sentences. The shortcut reflects the fact that proposition logic is insufficiently expressive to formalize this.

Indeed, the sentences in the preamble imply counting the number of true and false assertions. For many reasons not discussed here, Funmath views Booleans as numbers, subject to common arithmetic, which turns out advantageous for this kind of problems as well. We first make "one of the following assertions" more precise as "at least one of the following assertions", since interpretation with "exactly one" would make the second conjunct redundant, albeit this is clear only in the total context and gives the same end result (but that is hindsight).

A faithful translation of the preamble then proceeds as follows. The sentence "[at least] one assertion is true (false)" is in natural language shorthand for "the number of assertions that are true (false) is [at least] one". So, for the preamble,

$$\sum(\alpha,\beta)\geq 1 \quad\wedge\quad \sum(\neg\,\alpha,\neg\,\beta)\geq 1 \quad . \tag{1}$$

Equivalence with $\alpha\oplus\beta\equiv 1$ for Boolean α and β can be shown in many ways, e.g., the following head calculation in 3 steps using the formal rules of Funmath.

Generally, a family of sentences (such as α,β in problem (a) or α,β,γ in problem (b)) is a predicate, say P, and expressions like $\sum P\geq n$ or $\sum P=n$ as appropriate reflect the counting. The case $\sum P\geq 1$ is equivalent to $\exists\,P$, which is the formal rule for rewriting (1) in one step as $\exists\,(\alpha,\beta)\wedge\exists\,(\neg\,\alpha,\neg\,\beta)$. A second step using $\exists\,(p,q)\equiv p\vee q$ yields $(\alpha\vee\beta)\wedge(\neg\,\alpha\vee\neg\,\beta)$, which equals $\alpha\oplus\beta$.

The reader may wish to try this approach on problem (b) and then on some of the word problems in [11] or in the mathematical puzzles literature [9, 17].

From the classical "cleverness-oriented" problem solving point of view, faithful translation may seem overkill, but in the context of formal methods and textual specifications it can reduce errors. In view of the expressiveness and rich collection of formal rules in Funmath, the extra work need not be prohibitive.

3 Intermediate phases in formalizing informal statements

The preceding example already illustrated how to handle certain intermediate phases, but the problem statement itself was "static" and already logic-oriented.

Some interesting additional issues arise in the following problem from [7].

> A school has 1000 students and 1000 lockers, all in a row. They all start out closed. The first student walks down the line and opens each one. The second student closes the even numbered lockers. The third student approaches every third locker and changes its state. If it was open he closes it; if it was closed he opens it. The fourth student does the same to every fourth locker, and so on through 1000 students. To illustrate, the tenth locker is opened by the first student, closed by the second, reopened by the fifth, and then closed by the tenth. All the other students pass by the tenth locker, so it winds up being closed. How many lockers are open?

Here is the solution offered in [7].

> The n^{th} locker is opened or closed by student number k precisely when k divides n. So if student k changes locker n, so does student n/k. They cancel each other out. This always holds unless students k and n/k are precisely the same person. That is, $k = n/k$. The lockers that are exact squares will remain open. These are lockers 1, 4, 9, 16, 25, etc. How many of these are there in a row of 1000? You can go all the way up to $31 \times 31 = 961$, hence there are 31 lockers open.

In formalizing the problem statement, a first step is tightening the wording and removing examples. Here is the result.

> A school has 1000 students and 1000 lockers in a row, all initially closed. All students walk successively along the row, and the k^{th} student inverts the state of every k^{th} locker, that is: opens the locker if it was closed and vice versa. How many lockers are open in the end?

The formal equations reflecting the informal reasoning in the proposed solution, parametrized by the number of lockers N and the number of students K, are

$$\begin{aligned} \mathit{Answer} &= |\{n : 1\,..\,N \mid \mathit{Open}\, n\}| && \text{Legend: } |S| = \text{size of set } S \\ \mathit{Open}\, n &\equiv \mathsf{Odd}\,|\{k : 1\,..\,K \mid k \text{ divides } n\}| && \text{Legend: } \mathsf{Odd}\, m \equiv \text{number } m \text{ is odd} \end{aligned}$$

Elaborating yields a "nicer" expression for the answer, but this is not the issue.

The problem statement describes a procedure, the equations only the result. Classical mathematics cannot express the intermediate steps, but a procedural language can, and formal semantics allows deriving the equations formally.

A more faithful rendering of the procedure in the problem statement is

```
for k in 1..K do
    (for n in 1..N do if (k divides n) then inv (L n) fi od) od  .
```

Here `inv (L n)` (for "invert the state of locker `L n`") can be refined in many ways, for instance `L n := L n` $\oplus$ `1` if `L n` is defined as taking values in $0\,..\,1$. Program semantics allows calculating the final value of L (given that initially L is zero everywhere) and hence the answer $\sum L$. The calculation is facilitated by observing that the loops are interchangeable (even parallelization is possible).

In an introductory course, a scaled-down formal semantics can be used, kept simple by some restrictions on generality, as exemplified next.

Intermezzo: Microsemantics*, a scaled-down formal semantics* We show one of many forms for a simple program semantics presentable early in an introductory course when handling problems of this kind. It is assumed that one of the starting lessons was about substitution and instantiation, as in Gries and Schneider [11]. Substituting expression d for variable v in expression e is written $e\,[v := d]$, compacted as $e[^{v}_{d}$ (written e^{v}_{d} in [11]). As in [11], v and d may be tuples. The operator $[^{v}_{d}$ affects expressions only, the counterpart for commands is $\langle^{v}_{d}$.

In this example of a scaled-down semantics, the state s is the tuple of variables, in simple problems the one variable that is changed. A command c is a

function from states to states (*direct functional semantics*) defined recursively by axioms of the form $c\,s = e$ (instantiated $c\,d = e[^s_d]$). Here are axioms for the *basic* commands assignment $(v := e)$, composition $(c\ ;\ c')$ and selection.

$$\begin{aligned}
(v \texttt{ := } e)\,s &= s[^v_e \quad \text{or, as a nice pun, } (v \texttt{ := } e)\,s = s\,[v := e] \\
(c \texttt{ ; } c')\,s &= c'\,(c\,s) \\
(\texttt{if } b \texttt{ then } c \texttt{ else } c' \texttt{ fi})\,s &= b\,?\,c\,s \mathbin{!} c'\,s
\end{aligned}$$

The last right hand side is a *conditional expression* with axiom $(b\,?\,e_1 \mathbin{!} e_0) = e_b$. The following *derived* commands are expressed in terms of the basic ones.

$$\begin{aligned}
\texttt{skip} &= v \texttt{ := } v \\
\texttt{if } b \texttt{ then } c \texttt{ fi} &= \texttt{if } b \texttt{ then } c \texttt{ else skip fi} \\
\texttt{while } b \texttt{ do } c \texttt{ od} &= \texttt{if } b \texttt{ then (} c \texttt{ ; while } b \texttt{ do } c \texttt{ od) fi} \\
\texttt{for } i \texttt{ in } e \texttt{ .. } e' \texttt{ do } c \texttt{ od} &= i, i' \texttt{ := } e, e' \texttt{ ; while } i \leq i' \texttt{ do } c \texttt{ ; } i \texttt{ := } i \texttt{+1 od}
\end{aligned}$$

Finally, for arrays, $A\,i$ `:=` e is by definition shorthand for A `:=` $(i \mapsto e) \oslash A$. In Funmath, $d \mapsto e$ is a *maplet* as in Z [18], and $(f \oslash g)\,x = (x \in \mathcal{D}\,f)\,?\,f\,x \mathbin{!} g\,x$.

With these preliminaries, calculating the final value for L (after loop interchange) is left as an exercise for the reader. As this kind of approach is meant for an introductory course, elaboration should be done carefully and in detail (at least the first time) and at a pace that all students can follow.

Variants and ramifications An interesting item is the `k divides n` test, which is an indirect interpretation of "every k^{th} locker" in the problem statement. A more direct interpretation is that the k^{th} student always proceeds directly to the k^{th} following locker. This is reflected by the inner loop in the procedure below.

```
for k in 1..K do (n := k; while n ≤ N do inv (L n); n := n + k od) od
```

Some might find `(n := 0; while n + k ≤ N do n := n + k; inv (L n) od)` more stylish (I do). Anyway, now the loops are not interchangeable any more. Clearly the interplay between faithfulness of translation and simplicity of derivation provides enough sustenance for an entire course on specification and transformation.

As an aside, observe that this problem illustrates the reverse of program design, which starts from an abstract specification and results in a procedure. Here we start with a procedure and derive mathematical equations. In terms of axiomatic semantics, the solution involves calculating strongest postconditions, which also play an important role in the theory of reverse software engineering.

In the literature, the theoretical basis for postconditions is somewhat neglected as compared to preconditions (or anteconditions as we prefer to call them) and often presented as a footnote or afterthought. This is why we provide a more symmetric treatment in [6], where furthermore axiomatic semantics is not formulated via postulates but derived calculationally from *program equations*.

Again from the "cleverness-oriented" viewpoint, the procedural description and its analysis may seem superfluous, yet it shows how formal methods can attach "handles" to intermediate steps not expressible in standard mathematics.

A wealth of examples on *algorithmic problem solving* can be found in [2].

4 Using wide-scope domain-independent problems

Finally, we show that domain-independent problems can have a sufficiently wide scope to serve as a running example for all notational and reasoning issues in a complete introductory course on a mathematical basis for formal methods.

The chosen puzzle is designed as a brain-teaser and hence may appear somewhat artificial, but this is compensated by the fact that it was not designed at all for our purpose: it was proposed by Propp [16] in *Mathematical Horizons*. Moreover, its self-referential character is a good preparation for CS students.

Problem statement Table 2 is the test from [16]; we only renumbered the questions to range from 0 to 19. The author of the test further comments:

> The solution to [this] puzzle is unique; in some cases the knowledge that the solution is unique may actually give you a short-cut to finding the answer to a particular question, but it's possible to find the unique solution even without making use of the fact that the solution is unique. (Thanks to Andy Latto for bringing this subtlety to my attention.)
> I should mention that if you don't agree with me about the answer to #19, you will get a different solution to the puzzle than the one I had in mind. But I should also mention that if you don't agree with me about the answer to #19, you are just plain wrong. :-)

Formalization in Funmath Table 3 is the translation of Table 2 into mathematical equations. We directly encode the letters $\mathrm{A}\,..\,\mathrm{E}$ for the answers by $0\,..\,4$ to avoid the unnecessary clutter of explicit conversion mappings, so the answer to the test is a string $a : \square\, 20 \to \square\, 5$ satisfying this system of equations.

The notation is basic Funmath [4, 5] and hence needs no comment. We just mention $m\,..\,n = \{k : \mathbb{Z} \mid m \le k \le n\}$ and $\square\, n = 0\,..\,n-1$. A property of $\bigwedge$ is $m = \bigwedge (n : S \mid P\, n) \;\equiv\; P\, m \wedge \forall\, n : S\,.\, P\, n \Rightarrow m \le n$ for any subset S of $\mathbb{N}$ and predicate P on $\mathbb{N}$ with $\exists\, n : S\,.\, P\, n$. The choice operator $[]$ has axiom $\mathcal{R}\, f \neq \emptyset \Rightarrow [\,]\, f \in \mathcal{R}\, f$. Also, f^- is the generalized inverse of f, yielding inverse images iff they are unique [4], and $n\,\$\,a = \sum i : \mathcal{D}\, a\,.\, a\, i = n$ counts occurrences of n in a. The uniqueness operator ! is defined by $!\, P \;\equiv\; \forall\, (x, y) : (\mathcal{D}\, P)^2\,.\, P\, x \wedge P\, y \Rightarrow x = y$.

A few ad hoc operators: abs is the absolute value operator, and Evn etc. are appropriate predicates on $\mathbb{N}$ (i.e., their type is $\mathbb{N} \to \mathbb{B}$).

Note: we provide some extra information by stating here that no equation contains out-of-domain applications (e.g., a right-hand side outside $\square\, 5$). This is ensured by the designer of the test and captured in the formalization.

Calculating the solution(s) We shall use very few words; justifications are written between $\langle\ \rangle$, equation references between [ ]. Heuristic: we scan the list various times; first looking for equations yielding an answer by themselves, then extracting the maximum of information out of single equations, then in combination etc.. The numbering indicates how many answers are still left. Obviously, at the side we keep a running inventory of all answers found thus far, and occasionally we will show it.

Table 2. Self-Referential Aptitude Test

0. The first question whose answer is B is question
   (A) 0 (B) 1 (C) 2 (D) 3 (E) 4
1. The only two consecutive questions with identical answers are questions
   (A) 5 and 6 (B) 6 and 7 (C) 7 and 8 (D) 8 and 9 (E) 9 and 10
2. The number of questions with the answer E is
   (A) 0 (B) 1 (C) 2 (D) 3 (E) 4
3. The number of questions with the answer A is
   (A) 4 (B) 5 (C) 6 (D) 7 (E) 8
4. The answer to this question is the same as the answer to question
   (A) 0 (B) 1 (C) 2 (D) 3 (E) 4
5. The answer to question 16 is
   (A) C (B) D (C) E (D) none of the above (E) all of the above
6. Alphabetically, the answer to this question and the answer to the following one are
   (A) 4 apart (B) 3 apart (C) 2 apart (D) 1 apart (E) the same
7. The number of questions whose answers are vowels is
   (A) 4 (B) 5 (C) 6 (D) 7 (E)
8. The next question with the same answer as this one is question
   (A) 9 (B) 10 (C) 11 (D) 12 (E) 13
9. The answer to question 15 is
   (A) D (B) A (C) E (D) B (E) C
10. The number of questions preceding this one with the answer B is
    (A) 0 (B) 1 (C) 2 (D) 3 (E) 4
11. The number of questions whose answer is a consonant is
    (A) even (B) odd (C) a perfect square (D) a prime (E) divisible by 5
12. The only even-numbered problem with answer A is
    (A) 8 (B) 10 (C) 12 (D) 14 (E) 16
13. The number of questions with answer D is
    (A) 6 (B) 7 (C) 8 (D) 9 (E) 10
14. The answer to question 11 is
    (A) A (B) B (C) C (D) D (E) E
15. The answer to question 9 is
    (A) D (B) C (C) B (D) A (E) E
16. The answer to question 5 is
    (A) C (B) D (C) E (D) none of the above (E) all of the above
17. The number of questions with answer A equals the number of questions with answer
    (A) B (B) C (C) D (D) E (E) none of the above
18. The answer to this question is:
    (A) A (B) B (C) C (D) D (E) E
19. Standardized test is to intelligence as barometer is to
    (A) temperature (B) wind-velocity (C) latitude (D) longitude (E) all of the above

Table 3. Equations formalizing Table 2

$a\ \ 0 = \bigwedge i : \square\, 5 \mid a\, i = 1$	
$a\ \ 1 = [\!]\ (i : 5\,..\,9 \mid P\, i) - 5$ and $\exists\, P_{\in 5\,..\,9}$ and $!\, P$ where $P := i : \square\, 19\,.\, a\,(i+1) = a\, i$	
$a\ \ 2 = 4\,\$\,a$	
$a\ \ 3 = 0\,\$\,a - 4$	
$a\ \ 4 = (a_{<5})^-\ (a\, 4)$	
$a\ \ 5 = (3,3,0,1,2)\,(a\, 16)$	
$a\ \ 6 = 4 - \mathsf{abs}\,(a\, 7 - a\, 6)$	
$a\ \ 7 = (0\,\$\,a + 4\,\$\,a) - 4$	
$a\ \ 8 = \bigwedge i : \square\, 5 \mid a\,(i+9) = a\, 8$	
$a\ \ 9 = (3,0,4,1,2)^-\ (a\, 15)$	
$a\, 10 = 1\,\$\,a_{<10}$	
$a\, 11 = ((\mathsf{Evn}, \mathsf{Odd}, \mathsf{Sqr}, \mathsf{Prm}, \mathsf{Mof})^{\mathsf{T}}\ (1\,\$\,a + 2\,\$\,a + 3\,\$\,a))^-\ 1$	
$a\, 12 = ((a_{\mathsf{Evn}})^-\ 0 - 8)/2$	
$a\, 13 = 3\,\$\,a - 6$	
$a\, 14 = a\, 11$	
$a\, 15 = (3,2,1,0,4)^-\ (a\, 9)$	
$a\, 16 = (3,3,0,1,2)\,(a\, 5)$	
$a\, 17 = \forall\,(i : 1\,..\,4\,.\, 0\,\$\,a \neq i\,\$\,a)\,?\,4 \nmid ((\$\,a)\,\rceil\,(1\,..\,4))^-\ (0\,\$\,a) - 1$	
$a\, 18 = a\, 18$	
$a\, 19 = 4$	Question 19 is not mathematical, but asks an opinion.

With the numbering conventions as explained, here are the calculations.

20. [19] $\boldsymbol{a\, 19 = 4}$

19. [4] $a\, 4 = (a_{<5})^-\ (a\, 4)$
$\equiv$ ⟨Note⟩ $a\, 4 = (a_{<5})^-\ (a\, 4)\ \wedge\ a\, 4 \in \mathcal{D}\,(a_{<5})^-$
$\equiv$ ⟨Lemma $^-$⟩ $\boldsymbol{a\, 4 = 4}\ \wedge\ \forall\, i : \square\, 5\,.\, i \neq 4 \Rightarrow a\, i \neq a\, 4$
Lemma: $f\, j \in \mathcal{D}\, f^-\ \equiv\ j \in \mathcal{D}\, f \wedge \forall\, i : \mathcal{D}\, f\,.\, i \neq j \Rightarrow f\, i \neq f\, j$ (exercise)

18. [0] $a\, 0 = \bigwedge i : \square\, 5 \mid a\, i = 1$
$\equiv$ ⟨Prop. $\bigwedge$⟩ $a\,(a\, 0) = 1$ $[\alpha]$
$\wedge\ \ \forall\, i : \square\, 5\,.\, a\, i = 1 \Rightarrow a\, 0 \leq i$ $[\beta]$
$[\alpha]$ $\Rightarrow$ ⟨Leibniz⟩ $a\, 0 = 0 \Rightarrow a\, 0 = 1$
$\Rightarrow$ ⟨Leibniz⟩ $a\, 0 = 0 \Rightarrow 0 = 1$
$\equiv$ ⟨$p \Rightarrow 0\ \equiv\ \neg\, p$⟩ $a\, 0 \neq 0$ $[\alpha']$
$[\beta]$ $\Rightarrow$ ⟨Instantiate $i := 0$⟩ $a\, 0 = 1 \Rightarrow a\, 0 \leq 0$
$\Rightarrow$ ⟨Leibniz⟩ $a\, 0 = 1 \Rightarrow 1 \leq 0$
$\equiv$ ⟨$p \Rightarrow 0\ \equiv\ \neg\, p$⟩ $a\, 0 \neq 1$ $[\beta']$
$[\alpha]$ $\Rightarrow$ ⟨Leibniz⟩ $a\, 0 = 2 \Rightarrow a\, 2 = 1$
$\equiv$ ⟨[2]⟩ $a\, 0 = 0 \Rightarrow a\, 2 = 1 \wedge a\, 2 = 4\,\$\,a$
$\Rightarrow$ ⟨$a\, 4 = 4 \wedge a\, 19 = 4$⟩ $a\, 0 = 2 \Rightarrow a\, 2 = 1 \wedge a\, 2 \geq 2$
$\equiv$ ⟨$p \Rightarrow 0\ \equiv\ \neg\, p$⟩ $a\, 0 \neq 2$ $[\gamma']$
$[4]$ $\Rightarrow$ ⟨From step 19, Leibniz⟩ $\forall\, i : \square\, 4\,.\, a\, i \neq 4$
$\Rightarrow$ ⟨Instantiate $i := 0$⟩ $a\, 0 \neq 4$
$\Rightarrow$ ⟨$[\alpha', \beta', \gamma'], a\, 0 \in \square\, 5$⟩ $\boldsymbol{a\, 0 = 3}$

17. $[\alpha]$ $\Rightarrow$ ⟨$a\, 0 = 3$⟩ $\boldsymbol{a\, 3 = 1}$ $\Rightarrow$ ⟨[3]⟩ $0\,\$\,a = 5$

16. [9] $a\,9 = \text{`30412'}^{-}\,(a\,15)$
$\equiv \langle \text{Note} \rangle\ a\,9 = \text{`30412'}^{-}\,(a\,15) \wedge a\,15 \in \mathcal{D}\,\text{`30412'}^{-}$
$\Rightarrow \langle y \in \mathcal{D}\,f^{-} \Rightarrow x = f^{-}y \Rightarrow y = f\,x \rangle\ a\,15 = \text{`30412'}\,(a\,9) \quad [\delta]$
[15] $a\,15 = \text{`32104'}^{-}\,(a\,9)$, hence:
$a\,9 = \langle \text{Similarly} \rangle\ \text{`32104'}\,(a\,15)$
$= \langle [\delta] \rangle\ \text{`32104'}\,(\text{`30412'}\,(a\,9))$
$= \langle \text{Def. } \circ \rangle\ (\text{`32104'} \circ \text{`30412'})\,(a\,9)$
$= \langle \text{Calcul. } \circ \rangle\ \text{`03421'}\,(a\,9)$
The equation $x = \text{`03421'}\,x$ has just one solution, $x = 0$, so $\boldsymbol{a\,9 = 0}$.

15. $a\,15 = \langle [\delta], a\,9 = 0 \rangle\ 3$ Hence $\boldsymbol{a\,15 = 3}$

14. [16] $a\,16 = \text{`33012'}\,(a\ \ 5)$
$\Rightarrow \langle [5] \rangle\ a\,16 = \text{`33012'}^{2}\,(a\,16)$
$\Rightarrow \langle \text{Calcul. } \circ \rangle\ a\,16 = \text{`11330'}\,(a\,16)$
$\Rightarrow \langle \text{Solutions} \rangle\ a\,16 = 1 \vee a\,16 = 3$
[1] $\Rightarrow \exists\,(i : 5\,..\,9\,.\,a\,(i+1) = a\,i) \wedge !\,i : \square\,19\,.\,a\,(i+1) = a\,i$
$\Rightarrow \langle \text{Lemma} \rangle\ \forall\,i : \square\,19\,.\,i \notin 5\,..\,9 \Rightarrow a\,(i+1) \neq a\,i$
$\Rightarrow \langle \text{Instantiate } i := 15 \rangle\ a\,16 \neq a\,15$
$\Rightarrow \langle \text{Leibniz, } a\,15 = 3 \rangle\ a\,16 \neq 3$
$\Rightarrow \langle a\,16 = 1 \vee a\,16 = 3 \rangle\ \boldsymbol{a\,16 = 1}$
Lemma: $!\,P \Rightarrow X \subseteq \mathcal{D}\,P \Rightarrow \exists\,P_{\in X} \Rightarrow \forall\,x : \mathcal{D}\,P\,.\,x \notin X \Rightarrow \neg\,P\,x$

13. $a\,5 = \langle [5], a\,16 = 1 \rangle\ 3$ Hence $\boldsymbol{a\,5 = 3}$

12. [6] $a\,6 = 4 - \mathsf{abs}\,(a\,7 - a\,6)$
$\Rightarrow \langle \text{Arithmetic} \rangle\ a\,6 = 4 - (a\,7 - a\,6)$
$\vee\ a\,6 = 4 - (a\,6 - a\,7)$
$\Rightarrow \langle \text{Arithmetic} \rangle\ a\,7 = 4$
$\vee\ a\,7 = 2 \cdot (a\,6 - 2)$
$\Rightarrow \langle \text{Weaken} \rangle\ \mathsf{Evn}\,(a\,7)$
[7] $a\,7 = (0\,\$\,a + 4\,\$\,a) - 4$
$\Rightarrow \langle [2, 3], a\,3 = 1 \rangle\ a\,7 = a\,2 + 1 \quad [\sigma]$
$\Rightarrow \langle [\gamma''] a\,2 \geq 2 \rangle\ a\,7 \geq 3$
$\Rightarrow \langle \mathsf{Evn}\,(a\,7) \rangle\ \boldsymbol{a\,7 = 4}$

11. $[\sigma] \Rightarrow \langle a\,7 = 4 \rangle\ \boldsymbol{a\,2 = 3}$

We show the inventory thus far. Note: no more answers can be 4 (all used up).

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$a\,i$	3		3	1	4	3		4		0						3	1			4

10. [12] $a\,12 = ((a_{\mathsf{Evn}})^{-}\,0 - 8)/2$
$a\,12 = 0 \equiv \langle \text{Def. } ^{-} \rangle\ (a_{\mathsf{Evn}})^{-}\,0 = 12$
$\equiv \langle \text{Eq. } 12 \rangle\ a\,12 = 2$
$a\,12 = 1 \equiv \langle \text{Eq. } 12 \rangle\ (a_{\mathsf{Evn}})^{-}\,0 = 10$
$\equiv \langle \text{Def. } ^{-} \rangle\ a\,10 = 0$
$\equiv \langle [10], \mathrm{a}\,3 = 1 \rangle\ 0$
So $a\,12 \notin \{0, 1, 2\}$, hence $\boldsymbol{a\,12 = 3}$

9. [12] $a\,12 = ((a_{\mathsf{Evn}})^-\,0 - 8)/2$
$\quad\equiv\ \langle a\,12 = 3\rangle\ 3 = ((a_{\mathsf{Evn}})^-\,0 - 8)/2$
$\quad\equiv\ \langle$Arithmetic$\rangle\ (a_{\mathsf{Evn}})^-\,0 = 14$
$\quad\Rightarrow\ \langle$Def. $^-\rangle\ \boldsymbol{a\,14 = 0}$

8. [14] $a\,14 = a\,11$
$\quad\Rightarrow\ \langle \mathrm{a}\,14 = 0\rangle\ \boldsymbol{a\,11 = 0}$

7. [1] $\Rightarrow \exists\, i : 5\,..\,9\,.\,a\,i = a\,(i+1)$
$\quad\equiv\ \langle$Expand$\rangle\ a\,6 = a\,5\ \vee\ a\,7 = a\,6\ \vee\ a\,8 = a\,7\ \vee\ a\,9 = a\,8\ \vee\ a\,10 = a\,9$
$\quad\equiv\ \langle$Known$\rangle\ a\,6 = 3\ \vee\ 4 = a\,6\ \vee\ a\,8 = 4\ \vee\ 0 = a\,8\ \vee\ a\,10 = 0$
$\quad\equiv\ \langle$No more 4's$\rangle\ a\,6 = 3\ \vee\ a\,8 = 0\ \vee\ a\,10 = 0$
$\quad\equiv\ \langle (a_{\mathsf{Evn}})^-\,0 = 14\rangle\ \boldsymbol{a\,6 = 3}$

6. [1] $\Rightarrow a\,1 = [\!]\ (i : 5\,..\,9 \mid a\,(i+1) = a\,i) - 5 \wedge\ !\,i : \square\,19\,.\,a\,(i+1) = a\,i$
$\quad\Rightarrow\ \langle$Lemma, $a\,5 = a\,6\rangle\ \boldsymbol{a\,1 = 0}$
Lemma: $!\,P \Rightarrow X \subseteq \mathcal{D}\,P \Rightarrow \forall\, x : \mathcal{D}\,P\,.\,x \in \{x : X \mid P\,x\} \Rightarrow x = [\!]\ x : X \mid P\,x$

5. [8] $a\,8 = \bigwedge i : \square\,5 \mid a\,(i+9) = a\,8$
$\quad\Rightarrow\ \langle$Prop. $\bigwedge\rangle\ a\,(a\,8 + 9) = a\,8 \qquad [\kappa]$
$a\,8 = 0\ \Rightarrow \langle [12], a\,12 \neq 0\rangle\ 0$
$a\,8 = 1\ \Rightarrow \langle [\kappa, 10]\rangle\ a\,10 = 1 \wedge a\,10 = 2$
$a\,8 = 2\ \Rightarrow \langle [\kappa], 8.\rangle\ a\,11 = 2 \wedge a\,11 = 0$
So $a\,8 \notin \{0, 1, 2\}$, hence $\boldsymbol{a\,8 = 3}$

4. [10] $a\,10 = 1\,\$\,a_{<10}$
$\quad\Rightarrow \langle a_{<10} =$ '3031433430'$\rangle\ \boldsymbol{a\,10 = 1}$

We show once more the running inventory.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$a\,i$	3	0	3	1	4	3	3	4	3	0	1	0	3		0	3	1			4

3. Letting $b := a_{\notin\{13,17,18\}}$, earlier answers yield

i	0	1	2	3	4
$i\,\$\,b$	4	3	0	7	3

From step 17, $0\,\$\,a = 5$, so calculation (not shown) yields $0\,\$\,a_{\in\{13,17,18\}} = 1$.
[17] $\Rightarrow\ \langle a\,17 \neq 4, 0\,\$\,a = 5\rangle\ a\,17 = ((\$\,a)\rceil\,(1\,..\,4))^-\,5 - 1 \quad [\mu]$
$\quad\Rightarrow\ \langle$Note, prop.$^-\rangle\ \exists!\, i : 1\,..\,4\,.\,i\,\$\,a = 5$
$\quad\Rightarrow\ \langle$Arith.$\rangle\ 1\,\$\,a = 5$
$\quad\Rightarrow\ \langle [\mu], 1\,\$\,b = 3\rangle\ \boldsymbol{a\,17 = 0 \wedge a\,13 = a\,18 = 1}$

0. Result: $a =$ '30314334301031031014'.

This was an intermediate version, still needing some restyling, yet instructive also by its style imperfections. Although detailed, the formal derivation is not much larger than the informal statement. Deep mathematical problems like Fermat's "last theorem" will cause more expansion, "real-life" problems usually less.

As before, the point is *not* solving problems that cannot be solved without formal methods: the web contains informal solutions for this example. Even more: for a beginner, solving any but the smallest problem formally is harder, since it forces concentrating on solving the problem and learning the formalism.

The point is that the statement of this problem can be understood by any student without background in computing or other fields of engineering, yet

the formalization provides the opportunity for illustrating all notational issues relevant in modeling engineering systems and most formal rules needed to reason calculationally about them. Therefore, problems of this kind are ideal as running examples for any introductory course or textbook on formal methods.

5 Final remarks

An important observation Popular belief holds that formal methods are of little use in deriving formulas from informal statements, almost by definition. Yet, formal methods are especially valuable for this translation in the following way.

Precisely because informal statements are subject to interpretation, translation into formulas will generally yield different results — in fact, almost certainly if done from different viewpoints or, ideally, by different people. Formal calculation can then elucidate the relationship between the formalizations: equivalence, refinement, contradiction, hidden hypotheses etc., as the case may be.

Again, the problem in Section 4 can illustrate this: a literal translation of the statements in Table 2 will, for most of them, yield precursors of the equations in Table 3 rather than the equations themselves. Even more: problem solvers cited on the website mentioned in [16] have observed that some questions can be interpreted in emtirely different ways.

On language expressiveness Faithful translation requires a very expressive language, otherwise steps have to be skipped or even the link between the informal specification and the formulas is not easy to see. Since, with the current state of the art, automated tools still reflect to a large degree the restrictions imposed by the implementation or even some peculiar logic, they cause gaps.

An example of a tool-supported specification language that suffers less from this drawback than its peers is Lamport's TLA$^+$ [14], as it was designed with mathematical expression in mind. Therefore I particularly enjoy using this language and support tool (TLC) for the laboratory exercises in one of my courses.

Yet, a fully-fledged declarative language, designed with a preference for expressiveness over implementability, still offers advantages. Here is an example.

Informal specification: given a sequence of symbols, replace successive appearances of the same symbol (aptly called *stuttering* in the context of [14]) by a single appearance of that symbol. Sequences are defined as functions on natural numbers, e.g., of type $\mathbb{N} \to S$ for infinite sequences of elements of S.

Before continuing, the reader should express this in his/her preferred formalism. Even more interesting is letting students do this as a homework assignment.

Lamport's formal specification is the following. For any infinite sequences σ,

$$
\begin{array}{lcl}
\natural\sigma & \triangleq & \textsc{let}\ f[n \in Nat] \triangleq \textsc{if}\ n = 0\ \textsc{then}\ 0 \\
& & \qquad\qquad\qquad\qquad\qquad\qquad \textsc{else}\ \textsc{if}\ \sigma[n] = \sigma[n-1] \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{then}\ f[n-1] \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad \textsc{else}\ f[n-1] + 1 \\
& & \quad S \triangleq \{f[n] : n \in Nat\} \\
& & \textsc{in}\ \ [n \in S \mapsto \sigma[\textsc{choose}\ i \in Nat : f[i] = n]]
\end{array}
$$

I wanted to derive a formula from the informal specification that reflects the intuitive simplicity of the mapping. An essential feature that emerges from the statement is that the elements the sequence remain intact and in order; only the corresponding domain points are changed. This is exploited as follows.

Let us first provide some background. In Funmath, any function f satisfies $f = x : \mathcal{D}\, f \,.\, f\, x$ for new variable x (like $N = \lambda x.Nx$ for lambda terms), but also $f = \bigcup\!\!\cdot\, x : \mathcal{D}\, f \,.\, x \mapsto f\, x$, a merge of maplets. The effect of *merge* ($\bigcup\!\!\cdot$) for simple cases can be inferred from its use here, but its generic definition in [4] is more subtle and yields extra properties that make it extremely flexible, as illustrated by $f^- = \bigcup\!\!\cdot\, x : \mathcal{D}\, f \,.\, f\, x \mapsto x$ for *any* (not necessarily injective) function.

With this background, any sequence β can be written $\bigcup\!\!\cdot\, n : \mathcal{D}\, \beta \,.\, n \mapsto \beta\, n$. To transform this according to the specification, it suffices replacing the the domain point n to the left of $\mapsto$ by the number of times that a new "stutter" started before, which is $\sum k : \square\, n \,.\, \beta\,(k+1) \neq \beta\, k$. This yields the Funmath definition

$$\natural\, \beta \;=\; \bigcup\!\!\cdot\, n : \mathcal{D}\, \beta \,.\, \sum (k : \square\, n \,.\, \beta\,(k+1) \neq \beta\, k) \mapsto \beta\, n \tag{2}$$

for finite as well as infinite sequences. In a complete Funmath definition, equation (2) would be preceded by **def** $\natural : S^\omega \to S^\omega$ **with**, specifying the types.

Note that equation (2) is as succinct as the informal specification and easy to relate to the informal specification: immediately for those familiar with the formalism, and with the above derivation otherwise. Proving equivalence of equation (2) with Lamport's specification for infinite sequences, or with the semantics of a recursive Haskell program having the stated effect on finite sequences is a typical exam problem (subdivided into subproblems with helpful hints).

Educational issues In an ideal world, separation of concerns would be well-served by domain-independent problems making things easier on students. Yet, this does not guarantee a positive reception by all concerned. In courses, we found that some students react adversely, and a small minority (about 2 in 25) 'strongly asserts' (!) not being interested in puzzles or even in analogies with more tangible phenomena, but only wanting to do "real" applications and programming.

Taking such comments at face value is misleading. Indeed, when offered the choice between a 'theoretical' and an 'application' problem in a test, students mostly choose the former. In class exercises, they do less well on practical problems, and the mistakes or breaks in the answers show diffculties with combined concerns. Deeper probing via separate questionnaires strongly suggests that stating a preference for "real" problems is often only a pose, and that a dislike of mathematics is the real problem. Many prefer programming because of the immediate feedback from the computer and the chance to tinker until it works.

We conjecture that the growing supply of CS courses with just programming assignments on seemingly 'practical' but intellectually insignificant problems [19] degrades the ability to cope with the delay in gratification when doing math.

Yet, not taking the aforesaid comments for granted is also risky, because colleagues responsible for interpreting the questionnaires may well take them literally, especially if they are adverse to formal methods. In that case, the teacher faces the choice between serving the students or serving the administrators.

References

1. Vicki L. Almstrum, "Investigating Student Difficulties With Mathematical Logic", in: C. Neville Dean, Michael G. Hinchey, eds, *Teaching and Learning Formal Methods*, pp. 131–160. Academic Press (1996)
2. Roland Backhouse, *Algorithmic Problem Solving*. Lecture Notes, University of Nottingham (2005). On the web: `http://www.cs.nott.ac.uk/~rcb/G5AAPS/aps.ps`
3. Raymond T. Boute, *Funmath illustrated: a declarative formalism and application examples*. Declarative Systems Series No. 1, Computing Science Institute, University of Nijmegen (1993)
4. R. Boute, "Concrete Generic Functionals: Principles, Design and Applications", in: Jeremy Gibbons and Johan Jeuring, eds., *Generic Programming*, pp. 89–119, Kluwer (2003)
5. Raymond Boute, "Functional declarative language design and predicate calculus: a practical approach", *ACM TOPLAS, Vol. 27*, No. 5, pp. 988–1047 (Sep. 2005)
6. Raymond Boute, "Calculational semantics: deriving programming theories from equations by functional predicate calculus", to appear in *ACM TOPLAS* (2006)
7. Karl Dahlke, "Fun and Challenging Math Problems for the Young, and Young At Heart " `http://www.eklhad.net/funmath.html`
8. Edsger W. Dijkstra, "How Computing Science created a new mathematical style", *EWD 1073* (1990) `http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1073.PDF`
9. Martin Gardner, *My Best Mathematical and Logic Puzzles*. Dover (1994)
10. David Gries, *The Science of Programming*. Springer (1981, 5th printing 1989)
11. David Gries and Fred Schneider, *A Logical Approach to Discrete Math*. Springer (1993)
12. David Gries, "The need for education in useful formal logic", *IEEE Computer 29*, 4, pp. 29–30 (Apr. 1996)
13. Philip N. Johnson-Laird, (example problems in the psychological study of human reasoning), `http://www.princeton.edu/~psych/PsychSite/fac_phil.html`
14. Leslie Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley (2002)
15. Rex Page, *BESEME: Better Software Engineering through Mathematics Education*, project presentation `http://www.cs.ou.edu/~beseme/besemePres.pdf`
16. Jim Propp, "Self-Referential Aptitude Test", *Math Horizons, Vol. 12*, Feb. 2005, p. 35 (Feb. 2005) `http://www.maa.org/mathhorizons/volume/volume12.html`
17. Raymond Smullyan, *The Lady or the Tiger*. Random House (1992)
18. J. Mike Spivey, *The Z notation: A Reference Manual*. Prentice-Hall (1989).
19. Joel Spolsky, "The Perils of JavaSchools", in: *Joel on Software* (Dec. 29, 2005) `http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html`
20. Paul Taylor, *Practical Foundations of Mathematics* (second printing), No. 59 in *Cambridge Studies in Advanced Mathematics*, Cambridge University Press (2000); quotation from comment on chapter 1 in `http://www.dcs.qmul.ac.uk/~pt/Practical_Foundations/html/s10.html`
21. George B. Thomas, Maurice D. Weir, Joel Hass, Frank R. Giordano, *Thomas's Calculus* (11th. ed.). Addison Wesley (2004)
22. Frits Vaandrager, private communication (Feb. 2006)
23. Jeannette M. Wing, "Weaving Formal Methods into the Undergraduate Curriculum", *Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology (AMAST)* pp. 2–7 (May 2000) `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/calder/www/amast00.html`