Capítulo 15

A concepção estândar de prova e o Problema de Kant

André da Silva PORTO

O que é uma "demonstração matemática"? O objetivo deste pequeno texto será o de apresentar uma avaliação crítica da assim chamada "concepção estândar" do que seja uma "prova". Essa proposta, hegemônica na literatura, visa exatamente fornecer uma resposta para a pergunta que acabamos de formular no início de nosso texto.[1] Segundo essa concepção, um texto só poderia ser chamado de demonstração de uma certa proposição matemática se pudéssemos encontrar uma versão absolutamente decomposta e explicitada daquela demonstração, sua "versão completamente formalizada". Assim, o próprio caráter justificatório da demonstração ordinária original seria pensado como sendo de alguma forma dependente da existência daquele correlato formal. Ainda segundo essa concepção, os diversos textos ordinários que encontramos em nossos periódicos especializados e em nossos livros-texto só poderiam ser chamados de "demonstrações" de forma derivativa, secundária à obtenção, em geral deixada apenas no nível de uma possibilidade meramente contrafactual, de obtermos correspondentes formais daquelas provas ordinárias.

Como pano de fundo geral para nossa discussão, gostaríamos de propor um desafio que chamaremos de "o Problema de Kant": o de como deveríamos compreender a conexão entre um objeto ou evento físico qualquer (uma demonstração ou um cálculo) e as leis matemáticas necessárias que eles pretensamente "demonstrariam". Em uma formulação recente Michael Friedman escreve:

> Perhaps the most important problem facing interpretations of Kant's philosophy of mathematics, then, is to explain how, for Kant, sensibility and

[1] Também chamada de "*Derivation-Indicator View*" (Azzouni, 2004) e "*Formalizability Thesis*" (Feferman, 2012, p. 380)

> the imagination – faculties traditionally associated with the immediate apprehension of sensible particulars – can possibly yield truly universal and necessary knowledge. (Friedman, 2010, pp. 586-7)

No contexto da literatura atual sobre a filosofia kantiana, o Problema de Kant aparece geralmente associado às recentes investigações sobre o papel dos diagramas em demonstrações ordinárias (Friedman, 2012; Shabel, 2003). Num contexto assim, a questão diria respeito fundamentalmente a notações – sejam elas formais, ou não – que exemplificam o que elas mesmas denotam, ou exprimem. Um exemplo de notação assim seria os numerais em barras de Hilbert. Nessa notação, o numeral "I I I I I" instanciaria o número que ele mesmo denotaria. Não será esse o nosso enfoque com respeito ao problema de Kant. O que estaremos interessados nesse texto será no modo como várias abordagens – a clássica, a intuicionista e a de Wittgenstein – tratam o problema da relação entre uma *prova*, entendida como um objeto físico, uma inscrição, ou uma *derivação*, entendida como um processo estendido no tempo, e as leis matemáticas que esses objetos e processos físicos (ou mentais) visam a "demonstrar".

15.1 A "Concepção Estândar"

Comecemos nossa exposição pelo que chamamos de "explicação estândar" do que seja uma demonstração. Como antecipamos acima, o movimento elucidatório caraterístico dessa proposta envolve inicialmente o de dividir todas as demonstrações em dois grupos, as assim chamadas "provas formais" e as demonstrações ordinárias. Como já antecipamos, as demonstrações ordinárias, aquelas que encontramos em nossos livros-texto e periódicos matemáticos, mereceriam ser chamadas de "demonstrações" apenas em um sentido secundário, derivativo. Elas seriam assim apenas "indicadores" de suas respectivas provas formais (Azzouni, 2004, pp. 84-5). Em uma formulação alternativa recente, muito direta, lemos que

> According to the standard view, a mathematical statement is a theorem if and only if there is a formal derivation of that statement, or, more precisely, a suitable formal rendering thereof. (Avigad, 2019, p. 4)

Ainda segundo essa concepção estândar, uma demonstração ordinária seria como que uma espécie de "abreviação": uma abreviação de sua prova formal correspondente. Assim, para obtermos seu correlato formal – o único texto que realmente poderia assegurar o caráter de demonstração àquela formulação original – uma demonstração ordinária deveria ser completamente decomposta e explicitada, até que sua prova correspondente, completamente analisada, fosse

finalmente obtida e todas aquelas "abreviações" fossem inteiramente incluídas no texto final.

> ... the steps from one statement to the next whose justification may be evident to the human mathematician specializing in the subject matter of the proof but that require extensive filling in, in order to create a fully formal derivation. ... in practice, the expert human mathematician routinely calls on a repertoire of prior notions, methods and results from his memory to readily recognize the validity of the steps in question. ... they may also involve mathematics not explicitly present in the steps being filled in. (Feferman, 2012, p. 383)

Há aqui dois pontos que precisamos sublinhar, pois ele serão cruciais quando compararmos essas propostas mais tradicionais com o tratamento sugerido por Wittgenstein para a noção de "prova". Em primeiro lugar, se pensarmos em uma prova como um certo tipo de "justificação" de uma proposição matemática, então uma prova ordinária seria apenas uma "justificação parcial", uma espécie de "nota promissória" para sua correspondente formalizada. A prova formal, essa sim (e apenas ela) poderia fornecer a justificação final a proposição a ser demonstrada. Isso aconteceria porque, numa prova formal:

> [The inference rules] refer only to the outward structure of the formulas, not to their meaning, so that they could be applied by someone who knew nothing about mathematics, or by a machine. This has the consequence that there can never be any doubt as to what cases the rules of inference apply to, and thus the highest possible degree of exactness is obtained. (Gödel, 1995, p. 45)

Como também sublinhamos no início de nosso texto, essa verdadeira "nota promissória" seria normalmente deixada "à descoberto": a pretensa "prova formal" correspondente normalmente é deixada apenas como uma promessa, raras vezes levada à cabo.

> The problem is that formal derivations are very fragile objects and are few and far between. The derivation of even a straightforward result in elementary number theory can require thousands of inferences in axiomatic set theory, and a sequence of a thousand inferences in which a single one is incorrect is simply not a proof. How can even the most carefully written journal article indicate the existence of a long sequence of inferences without a single error? (Avigad, 2019, p. 4)

O segundo ponto importante que devemos enfatizar aqui é a aproximação, endêmica na literatura, entre "operações formais" e a ideia de um "maquinário",

de uma "operação puramente mecânica", algo que eventualmente poderia ser "rodado completamente às cegas", como em uma implementação levada a cabo por um computador, por exemplo.

> The notions of "formal system" and "mechanical operation" are intimately connected; either can be defined in terms or the other. If we should first define a mechanical operation directly (e.g., In terms of Turing machines), we would then define a "formal" system – one whose set of theorems could be generated by such a machine (that is to say, the machine grinds out all the theorems, one after another, but never grinds out a non-theorem). Alternatively (following the lines of Post), we can first define a formal system directly and define an operation to be "mechanical" or "recursive" if it is computable in some formal system. (Smullyan, 1961, p. 1)

Nosso interesse em sublinhar essa conexão entre "operação formal" e a ideia de um "processo que pudesse ser levado a cabo em um computador" parece introduzir uma segunda espécie de "justificação" para o caráter demonstrativo final da noção de "prova formal". A possibilidade de "mecanizar" completamente uma tal derivação pareceria trazer um certo "caráter objetivo" à nossa pretensão demonstrativa. Assim, introduziríamos um componente "quase-empírico" àquela justificação: uma prova formal estaria correta se, ao rodarmos aquela sequência formal em um computador, aquele aparato de fato acabasse produzindo como *output* o correspondente formal do teorema em questão, ao final do processo.[2]

É importante enfatizarmos aqui que este último componente "quase-empírico" da assim chamada "concepção estândar" jamais poderia ser aceito por um lógico-matemático clássico mais ortodoxo. Segundo aquela concepção tradicional, deveríamos fazer uma distinção radical entre "operações matemáticas" e "implementações empíricas" (dessas operações). Ainda segundo a proposta clássica, na melhor das hipóteses, poderíamos dizer que um certo processo empírico "instanciaria" alguma operação matemática. Mas, a identidade de uma operação assim estaria firmemente ancorada à extensão daquela operação, i.e., o conjunto das ênuplas ordenadas correspondente, uma "lista abstrata" de todos os argumentos possíveis daquela operação, juntamente com seus respectivos resultados, uma espécie de "tabuada abstrata", cuidadosamente depositada em um museu platônico ideal.

Para um clássico ortodoxo, nada do que possa vir a ser obtido em uma implementação mecânica poderia jamais ter qualquer impacto sobre esse reino puro de abstrações matemáticas. Ao contrário do que veremos no contexto intuicionista, qualquer ideia de um processo ao longo do tempo (seja ele físico ou mental)

[2] Ou, no caso dos atuais verificadores automáticos de provas, nossas propostas de derivação fossem finalmente "validadas".

"tornar verdadeira" uma proposição é severamente rechaçada. Wilfrid Hodges escreve:

> With any function $f(x, y)$ we think of someone taking the arguments (a, b) and turning them into the value $f(a, b)$. ... There is a cost in metaphors of this kind. Generally, what they say isn't literally true. We can't cause a mathematical structure A to be embedded in another structure B; in general, the most we can do is to describe an embedding. But very often we can't even do that, even when an embedding exists; it would take more than a lifetime to write out the description, or to compute what it is. Some embeddings can't be defined at all with the notions available to us. So, if these metaphors were taken literally, they would imply we have magical powers.... In spite of the attempts of set theorists to persuade us to think of functions as sets of ordered pairs, we persist in thinking of them as things that a person can do (i.e. has the power of doing). (Hodges, 2007, p. 7)

15.2 O Intuicionismo Sueco Contemporâneo

Um dos pontos mais surpreendentes sobre as formulações intuicionistas recentes, ligadas a Per Martin-Löf e a Dag Prawitz, talvez seja o quanto elas acabam se aproximando das propostas associadas à "concepção estândar", que acabamos de rapidamente delinear. Da mesma forma que na versão clássica, a concepção ordinária de uma "demonstração" seria subdividida em dois subgrupos distintos. Teríamos as demonstrações ditas "indiretas", ou "não-canônicas", e teríamos as provas diretas, canônicas. Novamente, da mesma forma que no caso da versão clássica, uma prova indireta só mereceria tal título por referência ao próprio processo de obtenção de uma prova direta, canônica:

> ... an indirect proof of a proposition is a method of proving it directly, that is, a method which yields a direct proof of the proposition as result. Thus, to know an indirect proof of a proposition is to know how to give a direct proof of it. (Martin-Löf, 1987, p. 413)

Por fim, ainda de forma análoga à versão tradicional, encontramos a ideia de uma "prova canônica" correspondente à cada uma das demonstrações não-canônicas (as demonstrações ordinárias), provas canônicas essas que, no mais das vezes, seriam deixadas apenas como "meras possibilidades teóricas", que em princípio "poderiam vir a ser obtidas". Até mesmo o recurso à ideia de uma "existência puramente abstrata", tão característica da abordagem clássica, é evocada para explicar esse apelo a essa modalidade deixada apenas "em princípio", a possibilidade de obtenção que ligaria uma demonstração indireta a seu (necessário) correspondente canônico:

> From the intuitionistic point of view, it is necessary that there exists in the abstract sense calculations of ... $10000000000000000000 = 10^{20}$... in order that it should be correct to assert $10^{10} \times 10^{10} = 10^{20}$...; but it is not necessary that these calculations be actually performed or that one of the proofs be constructed. (Prawitz, 1977, pp. 21-2)

Como também já havíamos antecipado acima, o tratamento da noção de "demonstração" pelo Intuicionismo sueco contemporâneo guarda importantes diferenças com respeito à versão estândar clássica. A noção de "demonstração" é subdividida em três conceitos distintos: os atos-de-prova, os objetos-prova e os traços-de-prova. Para um intuicionista, a noção de "demonstração" estaria inicialmente ligada a atos, os assim chamados "atos-de-prova". Esses atos-de-prova seriam processos inferenciais mentais de um sujeito (um matemático) em um determinado momento (uma certa sequência de raciocínios encadeados).

> My answer to the questions, what is a judgment? and, what is a proof of a judgment? is simply that a proof of a judgment is an act of knowing and that the judgment which it proves is the object of that act of knowing, that is, an object of knowledge. (Martin-Löf, 1987, p. 417)

Esses "atos-de-prova", quando bem-sucedidos, resultariam por sua vez, em "construções matemáticas", os "objetos-prova". Um ponto importante que devemos salientar aqui. Como a noção de "prova formal" tradicional, esses objetos-prova seriam entendidos, não como objetos físicos (textos), mas, sim, como "entidades abstratas", "*objetos matemáticos passíveis de serem operados e transformados, de forma inteiramente análoga a quaisquer outros objetos matemáticos*". (Sundholm, 1994, p. 121) E, conforme antecipamos acima, em forte contraste com o tratamento clássico, a proposta intuicionista fala diretamente nesses objetos-prova como sendo capazes de "*tornar verdadeiras*" proposições matemáticas:

> On my preferred constructivistic reading, the relevant truth-maker [for mathematical propositions] is, of course, the one introduced by Heyting, namely, that of a proof(-object) of the proposition in question. (Sundholm, 1993, p. 59)

Um último ponto merece ainda ser enfatizado aqui. A noção ordinária de "demonstração" – os textos que costumeiramente encontramos em livros e periódicos – não se confundiriam, nem com a noção de "ato-de-prova" (que seria puramente mental, subjetiva, "fugidia"), nem com a noção de "objeto-prova" (que seria puramente abstrata, uma "construção matemática como outra qualquer"). Os textos ordinários, inscrições no papel, seriam pensados como sendo apenas "resquícios" dos atos mentais correspondentes, os atos-de-prova (mentais):

> These acts, proofs in the subjective sense, when completed, have no further existence, but they may leave tracks or traces. These traces, or proofs in the objective sense, are what we find written down in mathematical texts and what may be used by other mathematicians to carry out proofs in the subjective sense for the same theorem. These are not the proofs that are at issue in the intuitionistic truth-maker conception [i.e., "proof-objects"]. (Sundholm, 1994, p. 121)

15.3 A Engenharia de Software

Deixemos nossa apresentação – sumária, é verdade – de alguns dos traços mais característicos do que poderíamos chamar de versão intuicionística contemporânea da noção de "demonstração" e retomemos nossa discussão mais geral sobre da concepção tradicional, clássica, daquela noção. A ideia central, de rebaixar a quase totalidade de nossas demonstrações ordinárias a um papel secundário – derivativo – e de insistir que nossos textos ordinários como sendo "demonstrações" apenas com respeito a uma noção fortemente idealizada de "prova" (as "provas completamente formalizadas"), tem sido criticada por um grande número de autores (Wang, 1955; Kreisel, 1985; Chateaubriand, 2005; Rav, 2008; Avigad, 2019).

Não vamos repassar aqui os aspectos fundamentais dessas críticas, com as quais, em geral, concordamos. Ao invés disso, faremos uma incursão – também rápida – em um contexto aparentemente muito distanciado da filosofia da matemática, a engenharia de software. Nossa ideia aqui é tentar levar a sério a aproximação entre a ideia de "prova formal" e "operação mecânica", como aquela proposta no trecho acima de Smullyan. Nesse novo contexto prático, encontraremos novamente três conceitos correlatos que desempenharam um papel decisivo nas concepções que vínhamos discutindo até agora. Estamos nos referindo inicialmente à ideia de "processo de formalização", como processo de "progressiva desabreviação de textos", em segundo lugar à noção de "provas ordinárias" (discursivas, diagramáticas ou mistas) e por fim, é claro, ao próprio conceito de "prova completamente formalizada". Aqui, nesse ramo da engenharia, como lá, na concepção clássica de "demonstração", esses três conceitos desempenham um papel semelhante e igualmente crucial. Porém, como veremos, num ramo eminentemente prático, como a engenharia de software, esses conceitos ganham uma fisionomia nova, completamente distinta daquela que encontramos na filosofia.

Comecemos com a ideia, frequentemente encontrada nos textos daquele ramo técnico, de ver o processo de construção de um programa como um processo de "refinamento gradual de uma especificação". Encontramos aqui um importante ponto inicial de conexão entre a concepção estândar de prova e a engenharia de

software. Aqui (como lá), encaramos o programa final como o resultado de um processo de explicitação e detalhamento – chamado de "refinamento" no contexto da engenharia de software. Assim, da mesma forma que uma prova ordinária, discursiva (ou diagramática), deveria ser analisada, passo a passo, até obtermos a prova formal correspondente, o programador também teria a tarefa de "refinar" a especificação inicial, normalmente aquela fornecida pelo próprio cliente que contrata os serviços dos engenheiros de software:

> A specification serves as a contract between a client who wants a computer to behave a certain way and a programmer who will program a computer to behave as desired. (Hehner, 2004, p. 41)

A partir da especificação inicial do cliente, normalmente expressa em linguagem natural, a tarefa do programador seria vista como a de um processo sucessivo de obtenção de especificações mais e mais refinadas, até atingirmos a "especificação final", o que chamamos normalmente de um "programa executável":

> A specifier should write the clearest, most understandable specification they can; a programmer's job is to refine it to obtain other specifications, the last of which is a program. (Hehner, 1999, p. 6)

Segundo essa abordagem, o mais importante não é propriamente o caráter formalizado de uma especificação. Cabe ao especificador apenas se concentrar em fornecer a "especificação mais clara e compreensível que puder" de cada estágio do processo. Segundo os preceitos da engenharia de software contemporânea, mais importante do que a formalização em si mesma (que pode ser precipitada), é mantermos cuidadosamente o caráter hierárquico, modular, dos sucessivos refinamentos, desde o seu início, no topo da hierarquia toda (com a especificação discursiva apresentada pelo cliente) até seu refinamento último (o programa executável, "formal"). Trata-se de conhecida uma estratégia no contexto da engenharia de software, a ideia de "dividir para conquistar!".

> A good top-down design avoids bugs in several ways. First, the clarity of structure and representation makes the precise statement of requirements and functions of the modules easier. Second, the partitioning and independence of modules avoids system bugs. Third, the suppression of detail makes flaws in the structure more apparent. Fourth, the design can be tested at each of its refinement steps, so testing can start earlier and focus on the proper level of detail at each step. (Brooks, 1995, p. 143)

A ideia é manter cuidadosamente o controle sobre o processo de refinamento através de uma hierarquização o mais transparente possível. Partimos de especificações mais gerais, onde a estrutura interna dos vários módulos e submódulos é

abstraída, de forma remover detalhamentos desnecessários e mesmo prejudiciais para aquele estágio do processo de refinamento. Dessa forma podemos melhor controlar, de cima para baixo, as especificações mais detalhadas a partir do que já tenha sido determinado nos níveis mais altos, mais abstratos.

> The two concepts [hierarchical types & abstract data types] are orthogonal – there may be hierarchies without hiding and hiding without hierarchies. Both concepts represent real advances in the art of building software. Each removes one more accidental difficulty from the process, allowing the designer to express the essence of his design without having to express large amounts of syntactic material that add no new information content. For both abstract types and hierarchical types, the result is to remove a higher-order sort of accidental difficulty and allow a higher-order expression of design. (Brooks, 1995, p. 189)

Finalmente, mesmo o resultado final – o "programa-executável" – é visto como sendo apenas mais uma especificação, conceitualmente homogênea com todas as anteriores. Assim, em um movimento argumentativo surpreendente, reminiscente da concepção clássica do que seja uma "função matemática", aqui, na engenharia de software, como lá, na matemática clássica, somos convidados a distinguir cuidadosamente o "programa", entendido como uma especificação abstrata do comportamento que uma máquina implementadora *deveria ter*, do comportamento efetivo que um certo aparato terá, de uma certa feita, em alguma execução específica daquele programa.

Estritamente falando, poderíamos até mesmo insistir que não faria sentido se falar em "rodar um programa", como normalmente fazemos. O que poderia ser realmente "rodado" seria sempre o "programa-texto", uma efetivação levada a cabo por uma máquina concreta, em uma determinada situação. O "programa-especificação", por si só, seria uma pura especificação ("abstrata") do como um processo assim (sua efetivação concreta) *deveria se dar*.

> A program is a description or specification of computer behavior. A computer executes a program by behaving according to the program, by satisfying the program. People often confuse programs with computer behavior. They talk about what a program "does"; of course, it just sits there on the page or screen; it is the computer that does something.... A program is not behavior, but a specification of behavior. (Hehner, 2004, p. 41)

Retomemos agora ao último dos elementos da proposta que chamamos de "concepção estândar de uma prova, a noção de "prova completamente formalizada". Poderíamos encontrar um equivalente para essa noção – onde todas as operações intermediárias, mesmo as mais simples, são cuidadosamente explicitadas – na ideia de uma "linguagem de máquina", uma linguagem do mais baixo

nível possível, onde todas as operações da máquina, mesmo as mais simples, são também cuidadosa e completamente especificadas. Assim, o próprio compilador, que geraria o programa executável, seria visto como uma parte "mecanizada", final, do processo de refinamento como um todo.

Podemos ir mesmo além da ideia de um "compilador em linguagem de máquina". A equivalência mais adequada da ideia de "prova formal completamente analisada" seria a de um nível de especificação ainda ulterior. Não somente as menores e menos importantes operações da máquina seriam previa e completamente especificadas, mas incluiríamos até mesmo o "*tracing*" dessas mesmas operações, o registro completo de todos os argumentos que servem de *input* para cada uma daquelas operações, seguidos de uma especificação completa e cuidadosa de seus efeitos resultantes.

Se juntarmos à linguagem de máquina a possibilidade de especificação abstrata da própria manipulação das várias estruturas de dados e valores de registros, obteremos o conhecido conceito, da engenharia de software, de uma "máquina abstrata":

> ... an abstract machine is nothing more than an abstraction of the concept of a physical computer. ...
>
> **Definition 1.1** (Abstract Machine) Assume that we are given a programming language, $\mathscr{L}$. An *abstract machine* for $\mathscr{L}$, denoted by $\mathscr{M}_{\mathscr{L}}$, is any set of data structures and algorithms which can perform the storage and execution of programs written in $\mathscr{L}$. (Gabbrielli & Martini, 2010, pp. 1-2)

Em um contexto assim, nenhum elemento seria deixado de fora de nossa especificação, nada seria "rodado", tudo estaria total e completamente especificado de antemão.

15.4 A Concepção Estândar vista à luz da Engenharia de Software

Deixemos a engenharia de software propriamente dita e vejamos agora como ficariam as propostas do que chamamos de "concepção estândar de prova", quando encaradas a partir do ponto de vista daquele ramo de engenharia. Como antecipamos, as propostas da concepção estândar são perfeitamente reconhecíveis no contexto da engenharia de software, mas parecem ir no sentido ... *precisamente contrário àquele preconizado pelas técnicas mais atualizadas e mais recomendáveis daquela disciplina técnica.* Assim, por exemplo, quando encarada do ponto de vista da engenharia de software, a ordem de precedência da concepção estândar estaria *completamente invertida!* Ao invés do refinamento de cima para baixo,

cuidadosamente hierarquizado e modularizado em uma sucessão progressiva de especificações, como ocorre naquela engenharia, a proposta estândar envolveria uma ênfase diametralmente oposta.

No contexto da concepção estândar, a recomendação seria pensar no processo de especificação como sendo determinado, de baixo para cima, pelas especificações mais detalhadas e menos transparente (as "provas formais completamente analisadas"). Ainda segundo aquela concepção, seriam aquelas especificações finais, normalmente deixadas no nível de uma mera "possibilidade teórica", que deveriam ser tomadas como determinantes da especificação inteira, de baixo para cima (e não o contrário, como é insistentemente preconizado na engenharia de software). Ora, uma proposta de uma inversão assim, enfatizando a programação feita diretamente em linguagens de mais baixo nível (as "linguagens de máquina") é uma abordagem encontrada apenas nos primórdios da computação, anterior até mesmo à introdução das primeiras ideias de "compilação automatizada de programas" e de "programas modularizados", como a famosa linguagem Fortran introduzida no início da década de 50 (Wexelblat, 1981, p. 9).

Ainda explorando os correspondentes da noção de "prova formal" no contexto da engenharia de software, a ênfase no *"tracing"* – a ideia clássica de que a identidade de uma função seria dada pela lista completa dos pares input/output – corresponderia, na engenharia de software contemporânea, novamente a um claro retrocesso. Ela corresponderia a insistirmos no assim chamado "*output testing*" como estratégia crucial, e decisiva, para verificação de um programa:

> The old technique was to make a program and then to subject it to a number of testcases where the answer was known; and when the testruns produced the correct result, this was taken as a sufficient ground for believing the program to be correct. But with growing sophistication, this assumption proved more and more to be unjustified. (Dijkstra E. , 1974, p. 610)
>
> We must conclude that exhaustive testing, even of a single component such as a multiplier, is entirely out of the question. (Dijkstra, Hoare, & Dahl, 1972, p. 4)

A ideia (clássica) de se fixar a identidade de uma função apenas pelos pares input/output (i.e., entender uma "função" como um conjunto de ênuplas ordenadas) parece tão idealizada ao ponto de soar ingênua. Ainda nas palavras de Edsger Dijkstra, um dos pais da moderna engenharia de software:

> Some years ago, a machine was installed on the premises of my University; in its documentation it was stated that it contained, among many other things, circuitry for the fixed-point multiplication of two 27-bit integers. A legitimate question seems to be: "Is this multiplier correct, is it performing according to the specifications?". The naive answer to this is: "Well, the

> number of different multiplications this multiplier is claimed to perform correctly is finite, viz. 2^{54}, so let us try them all". But, reasonable as this answer may seem, it is not, for although a single multiplication took only some tens of microseconds, the total time needed for this finite set of multiplications would add up to more than $10,000$ years! (Dijkstra, Hoare, & Dahl, 1972, p. 4)

Resumindo o que imaginamos ter obtido até aqui, nossa conclusão seria a de que, quando encaradas desde o ponto de vista da engenharia de software, as propostas da visão estândar pareceriam envolver uma *dose completamente inaceitável de idealização!* Dessa maneira, em nosso entendimento, o assim chamado "ceticismo com respeito à visão estândar" (Avigad, 2019, p. 5) não seria nada mais do que uma reação mais ou menos inevitável à essa proposta de idealização desenfreada. Desafios iniciais, imediatos, sobre como poderíamos implementar realmente as estranhas propostas da concepção estândar são normalmente deixadas na conta de uma "pura possibilidade teórica". Não deveríamos nos surpreender que problemas acabem por surgir:

> The probability of successful assessment decays exponentially with the length of the proof. Suppose there is a one percent chance of error in assessing the correctness of a step in an axiomatic derivation generated by some fallible means. Then the odds of correctly assessing the validity of a proof of one hundred steps is about 37% (roughly $1 = e$, where e is Euler's constant). At two hundred steps, the odds of success drop to less than 14%, and at four hundred steps, the odds drop to less than 2%. (Avigad, 2019, p. 4)

15.5 Uma Rápida Visita às Ideias de Wittgenstein

Apresentar as propostas de Wittgenstein para as noções de "prova" e de "regra matemática" é uma tarefa que fica grandemente facilitada quando encarada sob o pano de fundo de uma abordagem eminentemente prática, refratária a idealizações, como aquela adotada pela engenharia de software.

Comecemos com a própria noção, caraterística da engenharia de software, do que seja uma "especificação". Nos interessa aqui fundamentalmente o caráter normativo daquele conceito. Como diz Hehner no trecho que mencionamos acima, "um programa é uma descrição, ou especificação do comportamento do computador". Ora, há uma afinidade imediata aqui com o conhecido conceito de "regra" (matemática) de Wittgenstein. Da mesma maneira que lá, aqui também encaramos uma especificação fundamentalmente com sendo uma normatização, uma especificação introduzida para julgar aplicações empíricas, "implementações".

> The proposition proved by means of the proof serves as a rule — and so as a paradigm. For we go by the rule. (Wittgenstein, 1983, pp. 163, III, §28)
>
> Hence $4 + 1 = 5$ is now itself a rule, by which we judge proceedings. This rule is the result of a proceeding that we assume as *decisive* for the judgment of other proceedings. (Wittgenstein, 1983, pp. 318-9, VI, §16)

Em sua ênfase no caráter puramente normativo das regras Wittgenstein chega a propor até mesmo formulações puramente imperativas para leis matemáticas.

> Suppose we look at mathematical propositions as commandments, and even utter them as such? "Let 25^2 be 625". (Wittgenstein, 1983, pp. 271, V, §13)
>
> Can we imagine all mathematical propositions expressed in the imperative? For example: "Let 10×10 be 100". (Wittgenstein, 1983, pp. 276, V, §17)

Voltemos à formulação de Hehner acima: "um programa é uma *descrição*, ou especificação do comportamento do computador". Se entendermos os vários níveis especificatórios do processo de refinamento a partir de um ponto de vista estritamente normativo, então a própria utilização da palavra "descrição" na formulação de Hehner parece, de repente, um tanto fora de lugar ("*A program is a description or specification of computer behavior*"). Podemos *descrever*, é claro, o comportamento que uma máquina física teve em uma determinada ocasião. Mas, nesse caso, como enfatiza o próprio Hehner, não podemos nos furtar a possibilidade de que tal comportamento tenha envolvido algum tipo de falha ("a cabeça leitora do disco-rígido pode quebrar, o compilador pode ter um "bug..."). Mas, para aquele engenheiro da computação, uma especificação parece ser, não a *descrição* do comportamento de uma máquina concreta, de sua operação em uma dada ocasião, mas, sim (como certamente preferiria Wittgenstein), uma *prescrição* de como esse comportamento *deveria se dar*. Isso parece ficar claro trecho seguinte, quando o próprio Hehner acrescenta: "Um computador [em uma dada implementação] executa um programa comportando-se de acordo com o programa, satisfazendo o programa".

Encontramos aqui um novo e forte paralelo com conhecidas formulações devidas à Wittgenstein. Segundo o filósofo, uma proposição matemática jamais falaria sobre realidade alguma, seja uma realidade entendida como sendo "abstrata", seja ela tomada como sendo "empírica".

> In mathematics *everything* is algorithm, *nothing* meaning; even when it seems there's meaning, because we appear to be speaking *about* mathematical things in *words*. What we're really doing in that case is simply constructing an algorithm with those words. (Wittgenstein L. , 2005, pp. 494, §137)

> ... even if the proved mathematical proposition seems to point to a reality outside itself, still it is only the expression of acceptance of a new measure (of reality). (Wittgenstein L. , 1983, pp. 162, III §27)

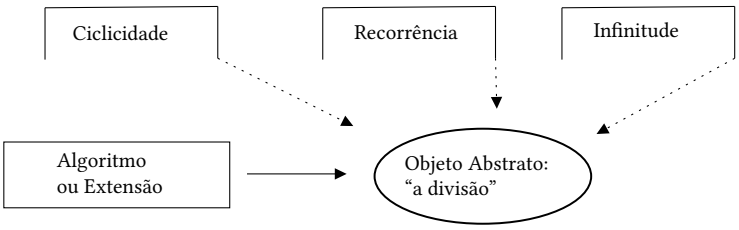
A matemática *não falaria sobre* nada, nem sobre objetos empíricos, tampouco sobre "objetos abstratos" (uma combinação de ideias – "objetos" "abstratos" – que Wittgenstein completamente repudia). A matemática seria composta apenas de especificações, de normatizações (as tais "regras" do filósofo[3]). Wittgenstein escreve, agora com respeito à geometria:

> Geometry isn't the science (natural science) of geometric planes, lines and points, as opposed, say, to some other science whose subject matter is gross physical lines, strips, surfaces, etc. and that states their properties. The connection between geometry and propositions of practical life, which are about strips, color boundaries, edges and corners, etc. doesn't consist in geometry's speaking of things similar to what these propositions speak of, although geometry does speak about ideal edges and corners, etc.; rather, it consists in the connection between these propositions and their grammar. Applied geometry is the grammar of statements about spatial objects. (Wittgenstein L. , 2005, pp. 391, §114) (Porto, 2012)

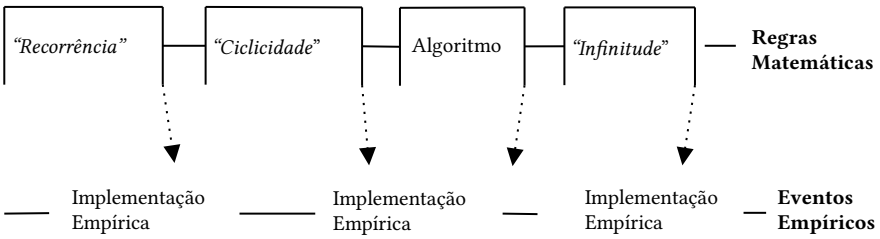
Vejamos agora outro componente fundamental para a filosofia da matemática de Wittgenstein, a ideia de um "holismo", i.e., de um antifundacionalismo com respeito à noção de "regras". Para o filósofo, como para o engenheiro de software, todos os vários níveis especificatórios, dos mais gerais (no topo de hierarquia), aos mais detalhados (em seus níveis mais baixos), devem ser homogeneamente encarados como sendo apenas especificações, "regras", no linguajar de Wittgenstein. Os vários níveis hierarquizados do processo de refinamento tomado como um todo especificariam – em conjunto – como deveriam se dar as várias implementações empíricas referentes àquelas especificações, como deveria se comportar as "máquinas físicas", propriamente ditas.

Tomemos o exemplo da longa discussão que Wittgenstein faz sobre o que seria "especificação" de uma operação aritmética simples, uma divisão recursiva como "$1 \div 7$" (Wittgenstein L. , 1979, pp. 182-4). Conforme uma visão mais tradicional, teríamos um "objeto abstrato" – "a divisão $1 \div 7$" fixado, ou bem por sua extensão (como sugeria o clássico), ou bem por sua definição, seu "método de geração" (como preferia o intuicionista). As várias propriedades, da "recorrência do resto", "recorrência do quociente", seriam todas vistas como descrições, como "predicações verdadeiras", daquelas "propriedades" àquele "objeto abstrato", a tal "função abstrata".

[3] Ou, talvez melhor ainda, "leis", um termo que retiraria completamente o caráter operacional que ainda permanece na palavra "regra". Esse termo, no entanto, não é empregado pelo filósofo.



Essa não seria, é claro, a proposta sugerida por Wittgenstein. Da mesma maneira que o engenheiro de software, o filósofo insiste em ver todos os níveis de especificação, tanto os mais gerais ("propriedades de alto nível", i.e., "ciclicidade do resto", "ciclicidade do quociente"), quanto os de mais baixo nível (asespecificações atômicas singulares, do tipo "$(1 \div 7)_{(1)} = 1$", ou "$(1 \div 7)_{(2)} = 4$") como determinações normativas, critérios que especificariam o que deveria acontecer em execuções empíricas daquela operação:



Nas palavras do filósofo, as próprias propriedades ("recursão no resto", "recursão do quociente") deveriam ser vistas como especificações ulteriores, de nível mais baixo, não de um "objeto abstrato", um conceito rejeitado pelo filósofo, mas, sim, de operações empíricas, implementações daquela operação levadas a cabo por um agente operador, em uma determinada situação:

> Here I am adopting a new criterion for seeing whether I divide this properly - and that is what is marked by the word "must". ...
>
> The question of recurrence is then a strictly geometrical question: the man will be persuaded that if he repeats this pattern here, there must be the same numeral repeated (A new criterion that he has done so-and-so) (Wittgenstein L. , 1975, pp. 129-30, Lect XII)

Cada nova "regra" é uma especificação ulterior, mais refinada, que qualquer implementação empírica daquela operação *deveria cumprir*, sob o risco, é claro, de termos de recusar aquele processo empírico como uma "implementação daquela especificação".

> One might also put it crudely by saying that mathematical propositions containing a certain symbol are rules for the use of that symbol, and that these symbols can then be used in non-mathematical statements. (Wittgenstein L. , 1975, pp. 33, Lect. III)

> ... only the group of rules *defines* the sense of our signs, and any alteration (e.g. supplementation) of the rules means an alteration of the sense. Just as we can't alter the marks of a concept without altering the concept itself. (Wittgenstein L. , 1975, pp. 182, §154)

Para finalizar, gostaríamos de registrar que, apesar de que todos os vários níveis especificatórios devam ser vistos homogeneamente como prescrições mais ou menos gerais sobre processos empíricos, do ponto de vista de seu "emprego prático", claramente, nem todas as determinações teriam igual importância. Ao contrário das propostas da concepção estândar, que vê os níveis mais baixo como, de alguma forma, "determinando, de baixo para cima, a verdade sobre os níveis superiores", Wittgenstein, como o engenheiro de software, encara o emprego das especificações como sendo, normalmente, "de cima para baixo".

Voltando novamente ao nosso exemplo da divisão periódica, pensemos no grande ganho operacional, técnico, que uma propriedade genérica como a da "recorrência do quociente" nos oferece na tarefa de julgar implementações empíricas daquela operação. Ao invés de termos de seguir, no nível mais baixo, um a um, os vários passos da "linguagem de máquina" (aqueles especificados pelo algoritmo de divisão), podemos adotar uma "visão se sobrevoo" e apenas verificar a recorrência daqueles dois ciclos, o ciclo do quociente ("333...") e do resto ("111...").

Retornemos ao trecho acima, onde Prawitz sugere que a "corretude" de uma equação (em "notação de alto nível", como "$10^{10} \times 10^{10} = 10^{20}$") seria de alguma forma dependente da "existência abstrata" de uma equação correspondente àquela, mas expressa em notação por sucessores, algo que só podemos indicar abreviadamente, é claro, como "$s(s(\ldots s(s(s(0)))) = s(s(\ldots s(s(s(0)))))$". Sobre uma notação muito semelhante à notação canônica "por sucessores" de Prawitz, a notação em barras de Hilbert, Wittgenstein escreve:

> ... could we also find out the truth of the proposition 7034174 + 6594321 = 13628495 by means of a proof carried out in the first notation [Bar notation]? – Is there such a proof of this proposition? – The answer is: no. (Wittgenstein L., 1983, pp. 145, III, §3) (RFM, Part III, §3, pg 145)

Em clara oposição às várias versões das propostas estândar e em absoluta consonância com o ponto de vista da engenharia de software (que insiste no movimento exatamente contrário), Wittgenstein afirma:

> A shortened procedure tells me what *ought* to come out with the unshortened one. (Instead of the other way around.) (Wittgenstein L., 1983, pp. 157, III, §18)

Como os engenheiros, Wittgenstein recusa a ideia de ver o nível mais baixo, da "linguagem de máquina", como sendo de alguma forma "mais fundamental" do que os outros:

> The danger here seems to be one of looking at the shortened procedure as a pale shadow of the unshortened one. (Wittgenstein L., 1983, pp. 157, III, §19)(RFM, Part III, §19, pg 157)

Wittgenstein rejeita as idealizações subjacentes às várias propostas estândar e, na companhia de seu colega engenheiro, acentua o emprego prático, o poder decisório em situações concretas de uso, que os vários níveis de especificações possam oferecer:

> ... in what sense is |||||||||||||||| the paradigm of a number? Consider how it can be used as such. (Wittgenstein L., 1983, pp. 147, III, §7)

Bibliografia

[1] Avigad, J. (2019). Reliability of Mathematical Inference. *Preprint*.

[2] Azzouni, J. (2004). The Derivation-Indicator View of Mathematical Practice. *Philosophia Mathematica, 12 (3)*, pp. 81-105.

[3] Brooks, F. (1995). *The Mythical Man-Month Essays on Software Engineering*. Boston: Addison Wesley.

[4] Chateaubriand, O. (2005). Proof and Proving. Em *Logical Forms - Part II*. São Paulo: Coleção CLE.

[5] Dijkstra, E. (1974). Programming as a discipline of mathematical nature. *American Mathematics Monthly, 81*, pp. 608-12. Fonte: E. W. Dijkstra Archive: http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD361.PDF

[6] Dijkstra, E., Hoare, C., & Dahl, O.-J. (1972). Notes on Structure Programming. Em E. Dijskstra, *Structured Programming* (pp. 1-82). Academic Press.

[7] Feferman, S. (2012). And so on...: Reasoning with infinite diagrams. *Synthese, 186 (1)*, pp. 371–386.

[8] Friedman, M. (2010). Synthetic History Reconsidered. Em M. Domski, & M. Dickson, *Discourse on a New Method: Reinvigorating the Marriage of History and Philosophy of Science* (pp. 571-813). Chicago: Open Court.

[9] Friedman, M. (2012). Kant on Geometry and Spatial Intuition. *Synthese, 186*, pp. 231-255.Gödel, K. (1995). *Collected Works* (Vol. III). Oxford: Oxford University Press.

[10] Gabbrielli, M., & Martini, S. (2010). *Programming Languages: Principles and Paradigms.* Dordrecht: Springer.

[11] Hehner, E. (1999). Specifications, Programs, and Total Correctness. *Science of Computer Programming, 34 (3)*, pp. 191-205.

[12] Hehner, E. (2004). *A Practical Theory of Programming.* Nova Iorque: Springer-Verlag.

[13] Hodges, W. (2007). Necessity in Mathematics I. Em *To appear as Proceedings of a conference on Necessity and Contingency.*

[14] Kreisel, G. (1985). Matthematical Logic: Tool and Object lesson for Science. *Synthese, 62*, pp. 139-151.

[15] Martin-Löf, P. (1987). Truth of a proposition, evidence of a judgment, validity of a proof. *Synthese, 73*, pp. 407-20.

[16] Porto, A. (2012). Wittgenstein on Mathematical Identities. *Disputatio, 4 (34).*

[17] Prawitz, D. (1977). The conflict between classical and intuitionistic logic. *Theoria, 63*, pp. 2-40.

[18] Rav, Y. (2008). The Axiomatic Method in Theory and in Practice. *Logique & Analyse, 202*, pp. 125-147.

[19] Shabel, L. (2003). *Mathematics in Kant's Critical Philosophy.* Nova Iorque: Routledge.

[20] Smullyan, R. (1961). *Theory of Formal Systems.* Princeton: Princeton University Press.

[21] Sundholm, G. (1993). Questions of Proof. *Manuscrito, 16(2)*, pp. 47-70.

[22] Sundholm, G. (1994). Existence, Proof and Truth-Making. *Topoi, 13*, pp. 117-26.

[23] Wang, H. (1955). On Formalization. *Mind, (64) 254*, pp. 226-238.

[24] Wexelblat, R. (1981). *History of the Programming Languages.* Nova Iorque: Academic Press.

[25] Wittgenstein, L. (1975). *Lectures no the Foundations of Mathematics, 1939.* Chicago: The Universtiy of Chicago Press.

[26] Wittgenstein, L. (1975). *Philosophical Remarks.* Oxford: Basil Blackwell.

[27] Wittgenstein, L. (1979). *Wittgenstein's Lectures, Cambridge, 1932-35.* Totowa: Rowan and Littlefield.

[28] Wittgenstein, L. (1983). *Remarks on the Foundations of Mathematics.* Cambridge: MIT Press.

[29] Wittgenstein, L. (2005). *The Big Typescript.* Malden: Blackwell Publishing.