9. Ciências da computação de Alan Turing: Uma viagem pessoal[1]

José Manuel Valença

O moderno cientista da computação, interessado na *correção* dos seus artefatos, navega num terreno construído sobre as respostas a algumas questões fundamentais. Primeiro ele/ela pode perguntar "O que é

1 Texto originalmente escrito em Inglês e traduzido pelo editor.

demonstrável, refutável ou decidível?", imediatamente seguido de "O que é inconsistente e o que pode ser validado?".

Mais ainda, é seguro dizer que, quando comparado com o âmbito do artefacto, ele/ela terá sobretudo que basear-se em respostas a outra questão "o que é que se pode aprender?". Finalmente, quando o interesse se estende à *segurança* do artefacto e à *privacidade* do utilizador, o cenário é complicado por questões adicionais, nomeadamente "O que é aleatório? O que é obscuro e à prova de fuga, no sentido da teoria da informação?".

Através de toda a paisagem da Ciência e Engenharia da Computação (CEC) podem detetar-se as pegadas de Turing, algumas desmaiadas, mas outras muito claras e incisivas. Nesta breve monografia tentarei delinear algumas dessas pegadas, escolhendo aquelas que eu acredito serem as mais significativas para a minha visão da CEC.

A questão que mais interessou Turing não foi nenhuma das anteriores mas antes

> O que é computável e, de entre essas 'coisas computáveis', o que é fazível?

É bem conhecido o papel de Turing (Turing 1936; Soare 2016) na construção de uma resposta pelo menos à primeira parte da questão. A fazibilidade das computações veio mais tarde e é o domínio da *Teoria da Complexidade*; a sua importância e o papel de Turing foram reconhecidos há algum tempo (Hartmanis 1994).

Algumas décadas antes, o início do Séc. XX viu o florescimento de novos fundamentos para o pensamento matemático: a axiomatização dos conjuntos de Zermelo-Fraenkel, construtivismo & intuicionismo e os problemas de decisão de Hilbert são alguns dos programas[2] desta era; para além do seu impacto na matemática como um todo, estes programas tiveram um efeito direto nos fundamentos da CEC; de facto, podemos dizer que estes "são" três quartos dos fundamentos da CEC, o restante quarto sendo, obviamente, a computabilidade.

2 Por "programa matemático" entendo uma linha de atividade que vai para além de estabelecer definições e teoremas e também inclui conjeturas, tentativas e erros (sobretudo estes) contribuindo para um conjunto coerente de crenças matemáticas.

No contexto da CEC, a linguagem dos quantificadores (LQ) tem um papel central. Em geral, a LQ é altamente relevante para qualquer programa matemático: como lógica de primeira ordem, segunda ordem ou ordem superior, em formalização clássica ou intuicionista, LQ é sempre uma candidata a linguagem matemática "standard".

Tome-se, por exemplo, ZFC, i.e. a axiomatização dos conjuntos de Zermelo-Fraenkel incluindo o axioma da escolha. ZFC é construída sobre a lógica de primeira ordem (LPO) e estende a Aritmética de Peano (AP). Portanto ZFC fornece um enquadramento muito genérico onde quer os objetos ("dados") quer as entidades de ordem superior (i.e. relações e algoritmos) da CEC podem ser formalizados.

Considere-se também o *Entscheidungsproblem* de Hilbert, o qual (em linguagem moderna) *procura construir um mecanismo universal capaz de transformar, com recursos finitos e incerteza negligenciável, uma frase f de LPO arbitrária numa resposta correta Sim/Não à questão "f é demonstrável em LPO?"*. Conjuntamente, estes dois programas fornecem a base inicial na discussão sobre "o que é demonstrável, refutável ou decidível".

A fazibilidade da indagação de Hilbert depende crucialmente do significado pretendido para o dito "mecanismo universal". Turing, ele próprio (Turing 1936), deu uma primeira resposta negativa: tomando "mecanismo universal" como sendo as suas "máquinas-a", equivalentes aos *procedimentos efetivos* de Gödel ou às *funções recursivas parciais* de Kleene, ele provou a existência de um máquina-a universal; a partir daí, e usando um argumento de diagonalização simples, ele construiu depois uma máquina-a cuja "propriedade de paragem" não pode ser decidida por nenhuma outra máquina-a.

Alguns anos antes, Gödel (Gödel 1931, Davis 2006) tinha demonstrado que nenhum sistema consistente de axiomas cujos teoremas possam ser listados por um procedimento efetivo (i.e. uma *axiomatização efectiva*) é capaz de demonstrar todas as verdades da aritmética dos números naturais; assim, Gödel não só refuta o programa de Hilbert, mas também nega a consistência de qualquer sistema formal efetivamente axiomatizado que fosse capaz de provar todos os teoremas da aritmética dos números naturais (uma hipotética extensão de ZFC, por exemplo).

No contexto dos Fundamentos da Matemática, estes são dois resultados negativos poderosos, ambos refutando o programa de Hilbert. No entanto, a CEC precisa, se não exatamente do programa de Hilbert, de algo muito semelhante: de facto, é essencial que se confie na correção dos artefactos da CEC (i.e. que se confie que eles fazem o que é suposto fazerem). Tradicionalmente, a confiança nos artefactos da CEC era procurada sobretudo através de testes; no entanto, recentemente, as provas formais estão a substituir os testes e a tornar-se o veículo preferencial para este propósito.

Com o construtivismo (Troelstra and van Dalen 1998), nomeadamente no contexto das *teorias de tipos construtivas* (Martin-Löf 1998), a correção dos artefactos da CEC tornou-se uma realidade: *assistentes de prova* como Isabelle, Coq e outros tornaram-se a norma em grandes projetos de CEC e a prova formal tornou-se um componente standard na Indústria do Software.

O papel de Turing aqui está bem documentado (Constable 2012). O aspeto central concerne a semântica da lógica intuicionista (LI), chamada de *semântica de Brower-Heyting-Kolmogorov (BHK)* mas mais conhecida na comunidade CEC como o *isomorfismo/analogia de Curry-Howard*. Essencialmente, este é um método que converte provas em programas e isto tem uma enorme importância, não só no nível fundamental, mas também no nível operacional. Primeiro, porque estabelece uma conexão direta entre "demonstrar" e "computar"[3]. Segundo, porque providencia uma forma consistente e rápida de assegurar construções computacionais corretas.

Nas últimas décadas podemos notar que "inconsistência", "validade" e "satisfazibilidade" se tornaram palavras familiares no vocabulário da CEC; de facto, elas estão na génese das ferramentas mais bem sucedidas ao dispor dos chamados *métodos formais* no desenvolvimento de software: os resolventes SAT, os resolventes SMT ("satisfazibilidade modulo teorias") e os "verificadores de modelos". Sistemas dinâmicos modernos muito grandes como controladores de tráfego aéreo, aviónica e dinâmica espacial altamente complexa, e, sobretudo, a microeletrónica altamente complexa que domina o nosso mundo, todos dependem destas ferramentas.

3 Isto teria agradado a Hilbert e Turing.

É claro, a *Teoria de Modelos*, tal como é usada hoje, é sobretudo uma criação de Gödel, Tarski (sobretudo) e Kripke; enquanto a *Teoria de Modelos "Finitos"* (TMF), da qual a maioria das ferramentas acima deriva, tem uma conexão muito mais forte com Turing. Um resultado standard de TMF (Libkin 2004) é o teorema de Trakhtenbrot que resumidamente diz que a validade de LPO na classe de modelos finitos é indecidível. Isto é crucial para a correção de resolventes SAT/SMT e este resultado vem essencialmente de Turing: a saber, o resultado é demonstrado mostrando que a classe de frases válidas em algum modelo finito não pode ser recursivamente enumerada.

O papel de Turing em Bletchley Park e na quebra do Código Enigma é evidentemente bem documentado; pelo menos tem sido objeto de vários livros e filmes de popularização. No entanto, afastando-nos da ficção, sabemos agora que, em meados dos anos 30, Jerzy Rozycki, Henryk Zygalski e Marian Rejewski usaram grupos simétricos[4] para cripto-analisar efetivamente o Código Enigma e propor a máquina (a "bomba") com a qual um ataque poderia ser levado a cabo. Também sabemos que Turing conhecia o seu trabalho e a "bomba" construída sob sua proposta foi um melhoramento significativo do design original.

O que é menos conhecido e mais interessante é o papel de Turing, não em criptografias antigas, mas nas dos dias modernos. A Criptografia Moderna requer *provas de segurança*[5] e, em consequência, existe investigação bem estabelecida no uso de assistentes de prova como fonte de confiança para esquemas e protocolos criptográficos. Também a *privacidade individual* se tornou uma preocupação criptográfica por fim tão relevante como a própria segurança do artefacto, e aqui é onde as máquinas-a de Turing se tornam relevantes.

A segurança demonstrável standard usa jogos para emular um esquema ou protocolo, e depois usa conceitos como "indistinguibilidade" ou "indiferenciabilidade" para comparar o comportamento do jogo como o de alguma forma de comportamento aleatório. Usualmente, há aqui um problema com uma mal definida noção de aleatoriedade. O uso da

4 Os serviços secretos britânicos, na altura, não acreditavam na matemática e preferiam a linguística.

5 Por exemplo, o corrente *NIST Post-Quantum Cryptography Standardization Process* requer, em cada submissão, uma prova completa de segurança PQ.

teoria completa da *aleatoriedade computacional* (Nies 2009, Downey e Hirschfeldt 2010) tornou-se um requisito moderno; ora tal teoria é um descendente direto das máquinas-a de Turing. De facto, sabemos que podemos modelar segurança e privacidade "à prova de fuga" com base, não só em aleatoriedade, mas também em "aleatoriedade indireta", os chamados oráculos "baixos" ou "K-triviais", os quais, de novo, são definidos com máquinas-a de Turing.

Poderíamos continuar nesta linha, citando problema atrás de problema, método atrás de método, até mesmo disciplinas inteiras de CEC, nos quais Turing deixou a sua inegável pegada. A sua importância não pode ser ignorada ou minimizada; graças a iniciativas como a deste volume, esperamos que ele nunca seja esquecido.

Referências

Constable, R. L. (2012). "On Building Constructive Formal Theories of Computation. Noting the Roles of Turing, Church, and Brouwer". In: 27th Annual IEEE Symposium on Logic in Computer Science, pp. 2–8. doi: 10.1109/LICS.2012.9.

Davis, Martin (2006). "The Incompleteness Theorem". In: Notices of the AMS 53.4, pp. 414–418.

Downey, R.G. and D.R. Hirschfeldt (2010). Algorithmic Randomness and Complexity. Theory and Applications of Computability. Springer New York. isbn:9780387684413. url: https://books.google.pt/books?id=FwIKhn4RYzYC.

Gödel, Kurt (1931). "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I." In: Kurt Gödel Collected works, Vol. I. (1986). Ed. by Solomon Feferman. ISBN 978-0195147209, pp. 144-195.

Hartmanis, Juris (1994). "Turing Award lecture: on computational complexity and the nature of computer science." In: Communications of the ACM.

Libkin, Leonid (2004). Elements of Finite Model Theory. Texts in Theoretical Computer Science. Springer-Verlag.

Martin-Löf, Per (1998). "An Intuitionistic Theory of Types". In: Twenty-Five Years of Construtive Type THeory. Ed. by Giovanni Sambin and Jan Smith.

Oxford Science Publications. Clarendon Press, Oxford, pp. 127-172.

Nies, Andre (2009). Computability and Randomness. Oxford University Press, Inc.

Soare, Robert Irving (2013). "Interactive Computing and Relativized Computability". In: Computability: Turing, Godel, Church, and Beyond. Ed. By Jack Copeland, Carl Posy, and Oron Shagrir. Cambridge Massachusetts, London: The MIT Press, pp. 203–260.

——(2016). Turing Computability: Theory and Applications. Theory and Applications of Computability. Springer.

Troelstra, A.S. and D van Dalen (1998). Construtivism in Mathematics: an Introduction. Vol. 1. Studies in Logic and the Foundations of Mathematics 121. North-Holland.

Turing, AM (1936). "On computable numbers, with an application to the Entscheidungsproblem". In: Proceedings of the London Mathematical Society 38.1931, pp. 173–198. url: http://classes.soe.ucsc.edu/cmps210/Winter11/Papers/turing-1936.pdf.