WIKIPEDIA

Turing completeness

In computability theory, a system of data-manipulation rules (such as a computer's instruction set, a programming language, or a cellular automaton) is said to be **Turing complete** or **computationally universal** if it can be used to simulate any Turing machine. This means that this system is able to recognize or decide other data-manipulation rule sets. Turing completeness is used as a way to express the power of such a data-manipulation rule set. Virtually all programming languages today are Turing complete. The concept is named after English mathematician and computer scientist Alan Turing.

A related concept is that of **Turing equivalence** – two computers P and Q are called equivalent if P can simulate Q and Q can simulate P. The Church–Turing thesis conjectures that any function whose values can be computed by an algorithm can be computed by a Turing machine, and therefore that if any real-world computer can simulate a Turing machine, it is Turing equivalent to a Turing machine. A universal Turing machine can be used to simulate any Turing machine and by extension the computational aspects of any possible real-world computer.[NB 1]

To show that something is Turing complete, it is enough to show that it can be used to simulate some Turing complete system. For example, an imperative language is Turing complete if it has conditional branching (e.g., "if" and "goto" statements, or a "branch if zero" instruction; see one instruction set computer) and the ability to change an arbitrary amount of memory (e.g., the ability to maintain an arbitrary number of data items). Of course, no physical system can have infinite memory; but if the limitation of finite memory is ignored, most programming languages are otherwise Turing complete.

Contents

Non-mathematical usage

In colloquial usage, the terms "Turing complete" or "Turing equivalent" are used to mean that any real-world general-purpose computer or computer language can approximately simulate the computational aspects of any other real-world general-purpose computer or computer language.

Real computers constructed so far can be functionally analyzed like a single-tape Turing machine (the "tape" corresponding to their memory); thus the associated mathematics can apply by abstracting their operation far enough. However, real computers have limited physical resources, so they are only linear bounded automaton complete. In contrast, a universal computer is defined as a device with a Turing complete instruction set, infinite memory, and infinite available time.

Formal definitions

In computability theory, several closely related terms are used to describe the computational power of a computational system (such as an abstract machine or programming language):

Turing completeness
: A computational system that can compute every Turing-computable function is called Turing-complete (or Turing-powerful). Alternatively, such a system is one that can simulate a universal Turing machine.

Turing equivalence
: A Turing-complete system is called Turing equivalent if every function it can compute is also Turing computable; i.e., it computes precisely the same class of functions as do Turing machines. Alternatively, a Turing-equivalent system is one that can simulate, and be simulated by, a universal Turing machine. (All known Turing-complete systems are Turing equivalent, which adds support to the Church–Turing thesis.)

(Computational) universality
: A system is called universal with respect to a class of systems if it can compute every function computable by systems in that class (or can simulate each of those systems). Typically, the term universality is tacitly used with respect to a Turing-complete class of systems. The term "weakly universal" is sometimes used to distinguish a system (e.g. a cellular automaton) whose universality is achieved only by modifying the standard definition of Turing machine so as to include input streams with infinitely many 1s.

History

Turing completeness is significant in that every real-world design for a computing device can be simulated by a universal Turing machine. The Church–Turing thesis states that this is a law of mathematics – that a universal Turing machine can, in principle, perform any calculation that any other programmable computer can. This says nothing about the effort needed to write the program, or the time it may take for the machine to perform the calculation, or any abilities the machine may possess that have nothing to do with computation.

Charles Babbage's analytical engine (1830s) would have been the first Turing-complete machine if it had been built at the time it was designed. Babbage appreciated that the machine was capable of great feats of calculation, including primitive logical reasoning, but he did not appreciate that no other machine could do better. From the 1830s until the 1940s, mechanical calculating machines such as adders and multipliers were built and improved, but they could not perform a conditional branch and therefore were not Turing complete.

In the late 19th century, Leopold Kronecker formulated notions of computability, defining primitive recursive functions. These functions can be calculated by rote computation, but they are not enough to make a universal computer, because the instructions which compute them do not allow for an infinite loop. In the early 20th century, David Hilbert led a program to axiomatize all of mathematics with precise axioms and precise logical rules of deduction which could be performed by a machine. Soon, it became clear that a small set of deduction rules are enough to produce the consequences of any set of axioms. These rules were proved by Kurt Gödel in 1930 to be enough to produce every theorem.

The actual notion of computation was isolated soon after, starting with Gödel's incompleteness theorem. This theorem showed that axiom systems were limited when reasoning about the computation which deduces their theorems. Church and Turing independently demonstrated that Hilbert's *Entscheidungsproblem* (decision problem) was unsolvable,[1] thus identifying the computational core of the incompleteness theorem. This work, along with Gödel's work on general recursive functions, established that there are sets of simple instructions, which, when put together, are able to produce any computation. The work of Gödel showed that the notion of computation is essentially unique.

In 1941 Konrad Zuse completed the Z3 (computer), the first working Turing-complete machine; this was the first digital computer in the modern sense.[2]

Computability theory

Computability theory characterizes problems as having, or not having, computational solutions. The first result of computability theory is that there exist problems for which it is impossible to predict what a (Turing-complete) system will do over an arbitrarily long time.

The classic example is the halting problem: create an algorithm which takes as input (a) a program in some Turing-complete language, and (b) some data to be fed to *that* program; and which determines whether the program, operating on the input, will eventually stop or will continue forever. It is trivial to create an algorithm that can do this for *some* inputs, but impossible

to do this in general. For any characteristic of the program's eventual output, it is impossible to determine whether this characteristic will hold.

This impossibility poses problems when analyzing real-world computer programs. For example, one cannot write a tool that entirely protects programmers from writing infinite loops, or protects users from supplying input that would cause infinite loops.

One can instead limit a program to executing only for a fixed period of time (timeout), or limit the power of flow control instructions (for example, providing only loops that iterate over the items of an existing array). However, another theorem shows that there are problems solvable by Turing-complete languages that cannot be solved by any language with only finite looping abilities (i.e., any language that guarantees every program will eventually finish to a halt). So any such language is not Turing complete. For example, a language in which programs are guaranteed to complete and halt cannot compute the computable function which is produced by Cantor's diagonal argument on all computable functions in that language.

Turing oracles

A computer with access to an infinite tape of data may be more powerful than a Turing machine: for instance, the tape might contain the solution to the halting problem, or some other Turing-undecidable problem. Such an infinite tape of data is called a Turing oracle. Even a Turing oracle with random data is not computable (with probability 1), since there are only countably many computations but uncountably many oracles. So a computer with a random Turing oracle can compute things that a Turing machine cannot.

Digital physics

All known laws of physics have consequences that are computable by a series of approximations on a digital computer. A hypothesis called digital physics states that this is no accident because the universe itself is computable on a universal Turing machine. This would imply that no computer more powerful than a universal Turing machine can be built physically.

Examples

The computational systems (algebras, calculi) that are discussed as Turing complete systems are those intended for studying theoretical computer science. They are intended to be as simple as possible, so that it would be easier to understand the limits of computation. Here are a few:

- Automata theory
- Formal grammar (language generators)
- Formal language (language recognizers)
- Lambda calculus
- Post–Turing machines
- Process calculus

Most programming languages, conventional and unconventional, are Turing-complete. This

includes:

- All general-purpose languages in wide use.
  - Procedural programming languages such as C, Pascal.
  - Object-oriented languages such as Java, Smalltalk or C#.
  - Multi-paradigm languages such as Ada, C++, Common Lisp, Object Pascal, Python, R.
- Most languages using less common paradigms
  - Functional languages such as Lisp and Haskell.
  - Logic programming languages such as Prolog.
  - general-purpose macro processor such as m4
  - Declarative languages such as XSLT.[3]
  - VHDL and other hardware description languages
  - Esoteric programming languages, a form of mathematical recreation in which programmers work out how to achieve basic programming constructs in an extremely difficult but mathematically Turing-equivalent language.

Some rewrite systems are Turing-complete.

Turing completeness is an abstract statement of ability, rather than a prescription of specific language features used to implement that ability. The features used to achieve Turing completeness can be quite different; Fortran systems would use loop constructs or possibly even goto statements to achieve repetition; Haskell and Prolog, lacking looping almost entirely, would use recursion. Most programming languages are describing computations on von Neumann architectures, which have memory (RAM and register) and a control unit. These two elements make this architecture Turing-complete. Even pure functional languages are Turing-complete.[4][NB 2]

Turing completeness in declarative SQL is implemented through recursive common table expressions.[5] Unsurprisingly, procedural extensions to SQL (PLSQL, etc.) are also Turing complete. This illustrates one reason why relatively powerful non-Turing-complete languages are rare: the more powerful the language is initially, the more complex are the tasks to which it is applied and the sooner its lack of completeness becomes perceived as a drawback, encouraging its extension until it is Turing complete.

The untyped lambda calculus is Turing-complete, but many typed lambda calculi, including System F, are not. The value of typed systems is based in their ability to represent most typical computer programs while detecting more errors.

Rule 110 and Conway's Game of Life, both cellular automata, are Turing complete.

Unintentional Turing completeness

Some games and other software are Turing-complete by accident.

Video games:

- *Dwarf Fortress*[6]
- *OpenTTD*
- *Minecraft*[7]
- *Minesweeper*[8]
- *LittleBigPlanet*[7]
- *Baba is You*
- *Factorio*[9]
- *Cities: Skylines*[10]
- *Opus Magnum*[11]

Card games:

- *Magic: The Gathering*[7][12]

Zero-person games (simulations):

- *Conway's Game of Life*[13][14]

Computational languages:

- HTML5 + CSS3[15][16]
- C++ Templates[17][18]

Computer hardware:

- x86 MOV instruction[19]

Non-Turing-complete languages

Many computational languages exist that are not Turing complete. One such example is the set of regular languages, which are generated by regular expressions and which are recognized by finite automata. A more powerful but still not Turing-complete extension of finite automata is the category of pushdown automata and context-free grammars, which are commonly used to generate parse trees in an initial stage of program compiling. Further examples include some of the early versions of the pixel shader languages embedded in Direct3D and OpenGL extensions.

In total functional programming languages, such as Charity and Epigram, all functions are total and must terminate. Charity uses a type system and control constructs based on category theory, whereas Epigram uses dependent types. The LOOP language is designed so that it computes only the functions that are primitive recursive. All of these compute proper subsets of the total computable functions, since the full set of total computable functions is not computably enumerable. Also, since all functions in these languages are total, algorithms for recursively enumerable sets cannot be written in these languages, in contrast with Turing machines.

Although (untyped) lambda calculus is Turing-complete, simply typed lambda calculus is not.

Data languages

The notion of Turing-completeness does not apply to languages such as XML, HTML, JSON, YAML and S-expressions, because they are typically used to represent structured data, not describe computation. These are sometimes referred to as markup languages, or more properly as "container languages" or "data description languages".

See also

- Algorithmic information theory
- Chomsky hierarchy
- Church–Turing thesis
- Computability theory
- Inner loop
- Loop (computing)
- Machine that always halts
- Rice's Theorem
- s_{mn} theorem
- Stephen Wolfram's *A New Kind of Science*
  - Principle of Computational Equivalence
- Structured program theorem
- Turing tarpit

Notes

1. A UTM cannot simulate non-computational aspects such as I/O.
2. The following book provides an introduction for computation models: Rauber, Thomas; Rünger, Gudula (2013). *Parallel programming: for multicore and cluster systems* (https://books.google.com/books?id=UbpAAAAAQBAJ&printsec=frontcover) (2nd ed.). Springer. ISBN 9783642378010.

References

1. Hodges, Andrew (1992) [1983], *Alan Turing: The Enigma*, London: Burnett Books, p. 111, ISBN 0-04-510060-8
2. Rojas, Raul (1998). "How to make Zuse's Z3 a universal computer" (https://www.researchgate.net/publication/3330654). *Annals of the History of Computing*. **20** (3): 51–54. doi:10.1109/85.707574 (https://doi.org/10.1109%2F85.707574).
3. Lyons, Bob (30 March 2001). "Universal Turing Machine in XSLT" (http://www.unidex.com/turing/utm.htm). *B2B Integration Solutions from Unidex*. Archived (https://web.archive.org/web/20110717162708/http://www.unidex.com/turing/utm.htm) from the original on 17 July 2011. Retrieved 5 July 2010.

4. Boyer, Robert S.; Moore, J. Strother (May 1983). *A Mechanical Proof of the Turing Completeness of Pure Lisp* (http://www.cs.utexas.edu/users/boyer/ftp/ics-reports/cmp37.pdf) (PDF) (Technical report). Institute for Computing Science, University of Texas at Austin. 37. Archived (https://web.archive.org/web/20170922044624/http://www.cs.utexas.edu/users/boyer/ftp/ics-reports/cmp37.pdf) (PDF) from the original on 22 September 2017.
5. Dfetter; Breinbaas (8 August 2011). "Cyclic Tag System" (https://wiki.postgresql.org/index.php?title=Cyclic_Tag_System&oldid=15106). *PostgreSQL wiki*. Retrieved 10 September 2014.
6. Cedotal, Andrew (16 April 2010). "Man Uses World's Most Difficult Computer Game to Create … A Working Turing Machine" (http://www.themarysue.com/dwarf-fortress-turing-machine-computer/). *The Mary Sue*. Archived (https://web.archive.org/web/20150627102458/http://www.themarysue.com/dwarf-fortress-turing-machine-computer/) from the original on 27 June 2015. Retrieved 2 June 2015.
7. Zwinkau, Andreas (20 October 2013). "Accidentally Turing-Complete" (https://web.archive.org/web/20150605121859/http://beza1e1.tuxen.de/articles/accidentally_turing_complete.html). *Andreas Zwinkau*. Archived from the original (http://beza1e1.tuxen.de/articles/accidentally_turing_complete.html) on 5 June 2015. Retrieved 2 June 2015.
8. Kaye, Richard (31 May 2007). "Infinite versions of minesweeper are Turing complete" (http://web.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf) (PDF). *Richard Kaye's Minesweeper Pages*. Archived (https://web.archive.org/web/20170202122129/http://web.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf) (PDF) from the original on 2 February 2017. Retrieved 14 March 2017.
9. "I made a programmable Turing-complete computer in Factorio" (https://www.reddit.com/r/factorio/comments/43giwy/i_made_a_programmable_turingcomplete_computer_in/). *Reddit*. 31 January 2016. Retrieved 2 February 2020.
10. Plunkett, Luke (16 July 2019). "Cities: Skylines Map Becomes A Poop-Powered Computer" (https://kotaku.com/cities-skylines-map-becomes-a-poop-powered-calculator-1836398063). *Kotaku*. Retrieved 16 July 2019.
11. Caldwell, Brendan (20 November 2017). "Opus Magnum player makes an alchemical computer" (https://www.rockpapershotgun.com/2017/11/20/opus-magnum-player-makes-computer/). *Rock Paper Shotgun*. Retrieved 23 September 2019.
12. Alex Churchill; Stella Biderman; Austin Herrick (2019). "Magic: The Gathering is Turing Complete". arXiv:1904.09828 (https://arxiv.org/abs/1904.09828) [cs.AI (https://arxiv.org/archive/cs.AI)].
13. Rendell, Paul (12 January 2005). "A Turing Machine in Conway's Game of Life" (http://rendell-attic.org/gol/tm.htm). *Rendell-Attic*. Archived (https://web.archive.org/web/20090708160241/http://rendell-attic.org/gol/tm.htm) from the original on 8 July 2009. Retrieved 22 June 2009.
14. Calcyman; Johnston, Nathaniel (16 June 2009). "Spartan universal computer-constructor" (http://www.conwaylife.com/w/index.php?title=Spartan_universal_computer-constructor&oldid=8792). *LifeWiki*. Retrieved 22 June 2009.
15. Fox-Epstein, Eli (26 August 2019), *Experimentations with Abstract Machines.* (https://github.com/elitheeli/stupid-machines), retrieved 26 August 2019
16. "CSS3 proven to be turing complete" (https://accodeing.com/blog/2015/css3-proven-to-be-turing-complete). *Accodeing to you*. 2015. Retrieved 26 August 2019.

17. Meyers, Scott (Scott Douglas) (2005). *Effective C++ : 55 specific ways to improve your programs and designs* (https://archive.org/details/effectivec55spec00meye) (3rd ed.). Upper Saddle River, NJ: Addison-Wesley. ISBN 0321334876. OCLC 60425273 (https://www.worldcat.org/oclc/60425273).
18. See History of TMP on Wikibooks
19. Dolan, Stephen. "mov is Turing-complete" (http://stedolan.net/research/mov.pdf) (PDF). *stedolan.net*. Retrieved 9 May 2019.

Further reading

- Brainerd, W.S.; Landweber, L.H. (1974). *Theory of Computation* (https://archive.org/details/theoryofcomputat00brai). Wiley. ISBN 0-471-09585-0.
- Giles, Jim (24 October 2007). "Simplest 'universal computer' wins student $25,000" (https://www.newscientist.com/article/dn12826-simplest-universal-computer-wins-student-25000.html). *New Scientist*.
- Herken, Rolf, ed. (1995). *The Universal Turing Machine: A Half-Century Survey*. Springer Verlag. ISBN 3-211-82637-8.
- Turing, A. M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem" (https://www.cs.ox.ac.uk/activities/ieg/e-library/sources/tp2-ie.pdf) (PDF). *Proceedings of the London Mathematical Society*. 2. **42**: 230–65. doi:10.1112/plms/s2-42.1.230 (https://doi.org/10.1112%2Fplms%2Fs2-42.1.230).
- Turing, A.M. (1938). "On Computable Numbers, with an Application to the Entscheidungsproblem: A correction". *Proceedings of the London Mathematical Society*. 2. **43**: 544–6. doi:10.1112/plms/s2-43.6.544 (https://doi.org/10.1112%2Fplms%2Fs2-43.6.544).

External links

- "Turing Complete" (https://c2.com/cgi/wiki?TuringComplete). *wiki.c2.com*.