WIKIPEDIA
The Free Encyclopedia

State (computer science)

In information technology and computer science, a system is described as **stateful** if it is designed to remember preceding events or user interactions;[1] the remembered information is called the **state** of the system.

The set of states a system can occupy is known as its state space. In a discrete system, the state space is countable and often finite. The system's internal behaviour or interaction with its environment consists of separately occurring individual actions or events, such as accepting input or producing output, that may or may not cause the system to change its state. Examples of such systems are digital logic circuits and components, automata and formal language, computer programs, and computers.

The output of a digital circuit or deterministic computer program at any time is completely determined by its current inputs and its state.[2]

Digital logic circuit state

Digital logic circuits can be divided into two types: combinational logic, whose output signals are dependent only on its present input signals, and sequential logic, whose outputs are a function of both the current inputs and the past history of inputs.[3] In sequential logic, information from past inputs is stored in electronic memory elements, such as flip-flops. The stored contents of these memory elements, at a given point in time, is collectively referred to as the circuit's *state* and contains all the information about the past to which the circuit has access.[4]

Since each binary memory element, such as a flip-flop, has only two possible states, *one* or *zero*, and there is a finite number of memory elements, a digital circuit has only a certain finite number of possible states. If N is the number of binary memory elements in the circuit, the maximum number of states a circuit can have is 2^N.

Program state

Similarly, a computer program stores data in variables, which represent storage locations in the computer's memory. The contents of these memory locations, at any given point in the program's execution, is called the program's *state*.[5][6][7]

A more specialized definition of state is used for computer programs that operate serially or sequentially on streams of data, such as parsers, firewalls, communication protocols and encryption. Serial programs operate on the incoming data characters or packets sequentially, one at a time. In some of these programs, information about previous data characters or packets received is stored in variables and used to affect the processing of the current character or packet. This is called a stateful protocol and the data carried over from the previous processing cycle is called the *state*. In others, the program has no information about the previous data stream and starts fresh with each data input; this is called a stateless protocol.

Imperative programming is a programming paradigm (way of designing a programming language) that describes computation in terms of the program state, and of the statements which change the program state. Changes of state are implicit, managed by the program runtime, so that a subroutine has visibility of the changes of state made by other parts of the program, known as side effects.

In declarative programming languages, the program describes the desired results and doesn't specify changes to the state directly.

In functional programming, state is usually represented with temporal logic as explicit variables that represent the program state at each step of a program execution: a state variable is passed as an input parameter of a state-transforming function, which returns the updated state as part of its return value. A pure functional subroutine only has visibility of changes of state represented by the state variables in its scope.

Finite state machines

The output of a sequential circuit or computer program at any time is completely determined by its current inputs and current state. Since each binary memory element has only two possible states, 0 or 1, the total number of different states a circuit can assume is finite, and fixed by the number of memory elements. If there are N binary memory elements, a digital circuit can have at most 2^N distinct states. The concept of state is formalized in an abstract mathematical model of computation called a finite state machine, used to design both sequential digital circuits and computer programs.

Examples

An example of an everyday device that has a state is a television set. To change the channel of a TV, the user usually presses a "channel up" or "channel down" button on the remote control, which sends a coded message to the set. In order to calculate the new channel that the user desires, the digital tuner in the television must have stored in it the number of the *current channel* it is on. It then adds one or subtracts one from this number to get the number for the new channel, and adjusts the TV to receive that channel. This new number is then stored as the *current channel*. Similarly, the television also stores a number that controls the level of volume produced by the speaker. Pressing the "volume up" or "volume down" buttons increments or decrements this number, setting a new level of volume. Both the *current channel* and *current volume* numbers are part of the TV's state. They are stored in non-volatile memory, which preserves the information when the TV is turned off, so when it is turned on again the TV will return to its previous station and volume level.

As another example, the state of a microprocessor is the contents of all the memory elements in it: the accumulators, storage registers, data caches, and flags. When computers such as laptops go into a hibernation mode to save energy by shutting down the processor, the state of the processor is stored on the computer's hard disk, so it can be restored when the computer comes out of hibernation, and the processor can take up operations where it left off.

See also

- Data (computing)

References

1. "What is stateless? - Definition from WhatIs.com" (http://whatis.techtarget.com/definition/stateless). *techtarget.com*.
2. Harris, David Money; Harris, Sarah L. (2007). *Digital Design and Computer Architecture* (https://books.google.com/books?id=5X7JV5-n0FIC&q=state&pg=PA115). USA: Morgan Kaufmann. p. 103. ISBN 978-0123704979.
3. Kaeslin, Hubert (2008). *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication* (https://books.google.com/books?id=gdRStcYgf2oC&q=state&pg=PA783). UK: Cambridge University Press. p. 735. ISBN 978-0521882675.
4. Srinath, N. K. (August 2005). *8085 Microprocessor: Programming and Interfacing* (https://books.google.com/books?id=FIYGSv3-C6IC&pg=PA46). Prentice-Hall of India Pvt. Ltd. p. 326. ISBN 978-8120327856. Retrieved 7 December 2012. "page 46"
5. Laplante, Philip A. (2000). *Dictionary of Computer Science, Engineering and Technology* (https://books.google.com/books?id=U1M3clUwCfEC&q=%22Program+state%22+%22&pg=PA24). USA: CRC Press. p. 466. ISBN 978-0849326912.
6. Misra, Jayadev (2001). *A Discipline of Multiprogramming: Programming Theory for Distributed Applications* (https://books.google.com/books?id=eZtxLnc3NbYC&q=%22Program+state%22+variables&pg=PA14). Springer. p. 14. ISBN 978-0387952062.
7. Prata, Stephen Prata (2004). *C Primer Plus, 5th Ed* (https://books.google.com/books?id=MYWQbufdVU4C&q=%22Program+state%22+variables&pg=PT113). Pearson Education. pp. 113–114. ISBN 978-0132713603.
