

CS375:

Logic and Theory of Computing

Fuhua (Frank) Cheng

Department of Computer Science
University of Kentucky

Table of Contents:

- **Week 1: Preliminaries** (set algebra, relations, functions) (read Chapters 1-4)
- **Weeks 2-5: Regular Languages, Finite Automata** (Chapter 11)
- **Weeks 6-8: Context-Free Languages, Pushdown Automata** (Chapters 12)
- **Weeks 9-11: Turing Machines** (Chapter 13)

Table of Contents (conti):

- **Weeks 12-13: Propositional Logic (Chapter 6), Predicate Logic (Chapter 7), Computational Logic (Chapter 9), Algebraic Structures (Chapter 10)**

8. Turing Machines and Equivalent Models — The Church-Turing Thesis

The **foundation of modern day computers** (**Turing machines**) originally was not developed for the construction of a computing device, but to **formalise (define/represent) algorithms**.

The Church-Turing Thesis – version 1

The Church-Turing Thesis:

A problem can be solved by an algorithm iff it can be solved by a Turing machine.

We know the definition of a Turing machine.

What is an algorithm?

The Church-Turing Thesis – version 1

What is an algorithm?

The Church-Turing Thesis – version 1

Let's see what the dictionaries say about algorithm:

“a rule for solving a mathematical problem in a finite number of steps”

...Chambers' Dictionary

“process or rules for (esp. machine) calculation”

... Oxford dictionary

The Church-Turing Thesis – version 1

Are these definitions
precise and unambiguous?
simple?
general?

Besides, why would they relate Turing
machines to algorithms?

The Church-Turing Thesis – version 1

A little history ...

The Church-Turing Thesis – version 1

- In the late 19th Century, a problem exercising mathematicians was one of those posed by Hilbert:

“Is there a Universal Algorithm which can solve all Mathematical problems?”

... attempts to find one failed ...

...so perhaps there isn't one ...

...can we prove there is no universal algorithm?

...we need to be able to define an algorithm precisely so as to prove properties of algorithms

Formalism for algorithms

By the 1930s the emphasis was on formalising algorithms

Alan Turing, at Cambridge, devised an abstract machine now called a Turing Machine to define/represent algorithms

Alonso Church, at Princeton, devised the Lambda Calculus which formalises algorithms as functions

(more in the course 'computability and complexity')

Formalism for algorithms

neither knew of the other's work in progress ...both published in 1936

the demonstrated equivalence of their formalisms strengthened both their claims to validity, expressed as the Church-Turing Thesis

The Church-Turing Thesis – version 1

The Church-Turing Thesis:

A problem can be solved by an algorithm iff it can be solved by a Turing machine.

Turing machines implement algorithms

all algorithmically solvable problems can be solved by a Turing Machine

The Church-Turing Thesis – version 1

The Church-Turing Thesis:

A problem can be solved by an algorithm iff it can be solved by a Turing machine.

“a function is computable iff it can be solved by a Turing Machine”

“an algorithm is what a Turing Machine implements”

The Church-Turing Thesis – version 1

This is what an **algorithm** should really be defined:

An algorithm is a *step-by-step procedure* to solve logical and mathematical problems

with *instructions* to define/specify how each step should be carried out (or, implemented)

The Church-Turing Thesis – version 1

This is what an **algorithm** should really be defined:

The steps and instructions might not be unique, but must be precise

i.e., if the same procedure is repeated again, you should get exactly the same result

The Church-Turing Thesis – version 1

Does the ***Church-Turing Thesis*** make clear sense now?

The Church-Turing Thesis:

A problem can be solved by an algorithm iff it can be solved by a Turing machine.

The Church-Turing Thesis – version 1

The Church-Turing Thesis:

A problem can be solved by an algorithm iff it can be solved by a Turing machine.


First, can the function of a TM be described as an algorithm?

If the answer is YES, then this direction of the Church-Turing Thesis is true.

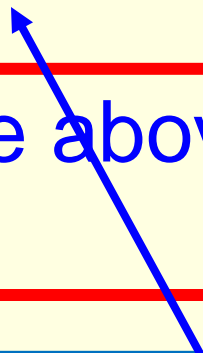
The Church-Turing Thesis – version 1

The Church-Turing Thesis:

A problem can be solved by an algorithm iff it can be solved by a Turing machine.



Can any algorithm (defined the above way) be implemented by a TM?



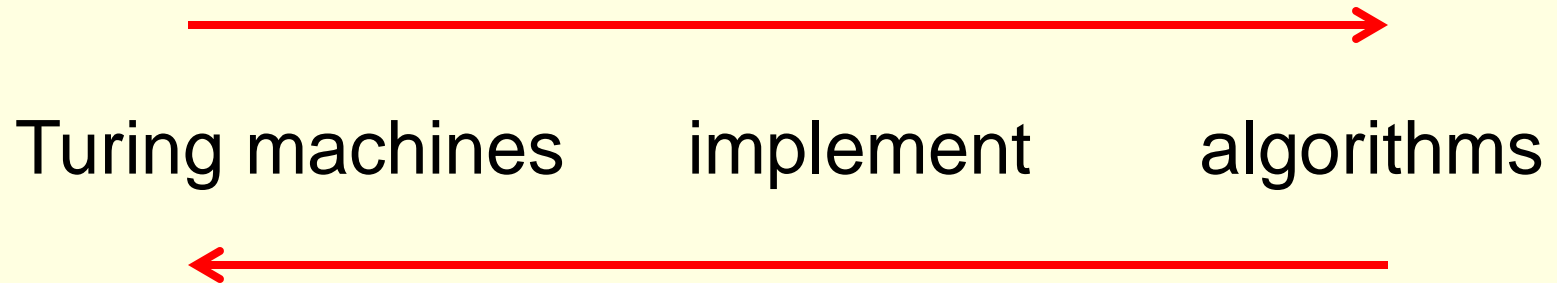
If the answer is YES, then this direction of the Church-Turing Thesis is true.

The Church-Turing Thesis – version 1

The Church-Turing Thesis:

A problem can be solved by an algorithm iff it can be solved by a Turing machine.

So we would have:



The Church-Turing Thesis – version 1

So, eventually, this is all we have to answer:

Can any algorithm (defined the above way)
be implemented by a TM?

The Church-Turing Thesis – version 1

Important information here

Thesis not Theorem

because we cannot prove this...
with a counter example we could disprove it
(but this has not been done).

we can show **supporting evidence** for the validity of the thesis

The Church-Turing Thesis – version 2

The Church-Turing Thesis:

Anything that is intuitively computable can be computed by a Turing machine.

Intuitive computable: algorithmically solvable
(a detailed algorithm for manual calculation can be developed)

The Church-Turing Thesis – version 2

It is a *thesis* rather than a *theorem* because it relates the *informal* notion of intuitively computable to the *formal* notion of a Turing machine.

The Church-Turing Thesis – version 2

Computational Models

A *computational model* is a **characterization** of a **computing process** that describes the **form of a program** and describes how the instructions are executed.

Example. The Turing machine computational model describes the **form of TM instructions** and **how to execute them**.

Example. If X is a programming language, the **X computational model** describes the **form of a program** and **how each instruction is executed**.

The Church-Turing Thesis – version 2

Equivalence of Computational Models

Two computational models are *equivalent* in power if they solve the same class of problems.

Any piece of data for a program can be represented by **a string of symbols** and any string of symbols can be represented by **a natural number**.

So even though computational models may process different kinds of data, they can still be compared with respect to how they process natural numbers.

References:

<https://plato.stanford.edu/entries/church-turing/>

<http://www.doc.ic.ac.uk/~mrc/Computability%20&%20Complexity/Lectures/C240Lecture2.pdf>

End of Turing Machines III