SE 3310b

Theoretical Foundations of Software Engineering

Decidability and the Church-Turing Thesis

Aleksander Essex

Western Engineering

Church-Turing Thesis

Effective Method

To solve the *Entscheidungsproblem* (Decision Problem), Turing offered an intuitive notion of what he called an ”effective method” for computing:

1. Takes a finite number of steps
2. Always produces a result
3. Always produces a correct answer
4. Could in principle be done by a human (with enough time, pencils and paper)
5. Only need to follow instructions, no ingenuity or creativity required

Definition of an Algorithm

An ”effective method” is an intuitive definition of an *algorithm*. Additionally,

- A function is ”effectively calculable” if there is an effective method (i.e., algorithm) to do so.
- A function is ”computable function” is there is a Turing Machine that can compute it.

Church-Turing Thesis

The Church-Turing thesis is the result of Alonzo Church and Alan Turing's efforts to capture the notion of computation and its real-world limits.

Definition 1 (Church-Turing Thesis)**.**

Every effectively calculable function is a computable function.

In other words, a problem can be solved by an algorithm if and only if it can be solved by a Turing machine.

Summary

In summary the Church-Turing thesis is basically saying that Turing machines (or equivalently Church's Lambda Calculus) can implement any algorithm.

This, however is a *thesis*, not a theorem. It can be disproved by a counter example. Arguments *for* include:

- Huge set of known Turing computable functions, no known counter examples
- Equivalence with other models (e.g., Lamba calculus, cellular automata etc)

So when we say a Turing machine can "compute" a function, what do we mean exactly?

Decidable vs. Recognizable

Classifying languages

When a Turing machine is run given a particular string, three outcomes are possible:

1. Accept and halt
2. Reject and halt
3. Loop forever (i.e., never halts)

Telling the difference between a machine that halts (but is just taking a long time) from one that never halts seems like a useful distinction. Therefore we define two classes of languages: one for which the Turing machine always halts (i.e., always accepts or rejects) from ones that may loop on certain inputs.

Recognizing vs. Deciding

Let L be a language and M be a Turing machine.

- **Recognizable Language**: We say M *recognizes* L if M halts and accepts all strings $s \in L$. Note we don't place any condition on what the Turing machine should do if it encounteres a string $s \notin L$.
- **Decidable Language**: We say M *decides* L if M if M halts and accepts all strings in $s \in L$, and halts and rejects all strings $s \notin L$.

Thus a Turing machine that decides a language always halts, whereas a Turing machine that recognizes a language must only halt for strings in the langauge. Thus it follows that all decidable languages are recognizable languages.

Western Engineering

Recognizable Languages

Definition 2 (Turing Recognizable)**.**

A language is called *Turing recognizable* if some Turing machine recognizes it.

The class of Turing recognizable langauges is sometimes called the *recursively enumerable* languages.

Decidable Languages

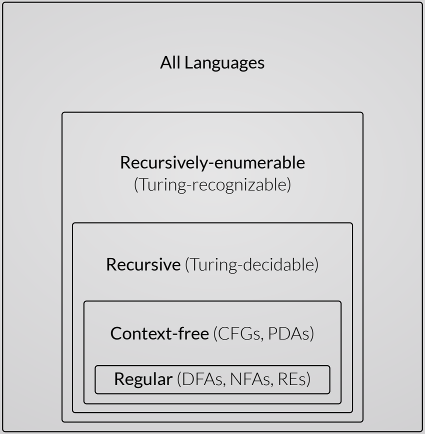
A Turing machine M that always halts given any input is called a *decider*. Because it always halts, we say it *decides* whether a string is in the language or not.

Definition 3 (Turing Decidable)**.**

A language is called *Turing decidable* (i.e, *decidable*) if some Turing machine decides it.

The class of Turing-decidable languages is sometimes called the *recursive* languages.

Language Hierarcy



Note: Context-sensitive langauges not shown.

Western Engineering