Overview of Calculability

Based on the slides of Maurizio Gabbrielli

Can we get rid of mistakes?

- Can we detect all possible errors, i.e. can there be a compiler that detects all possible errors that will occur at run-time?

An example

- Two Java programs:
  - A syntactically correct one:

```
int k = 1;
while (k == k) {}
System.out.println(k);
```

  - One syntactically incorrect:

```
int k = 1;
while (true) {}
System.out.println(k);
```

Because

- The Java specification says:

  It is a compile-time error if a statement cannot be executed because it is unreachable. Every Java compiler must carry out the conservative flow analysis specified here to make sure all statements are reachable.
- In the first program the compiler does not notice that k = k is equivalent to True
- In the second program Yes! (and then realizes that System.Out.println (k) will never be reached)

The Halting Problem

- Suppose we have the application Halt(P,X) that checks if a program terminates on X
  - Halt(P,X) = True if P(X) terminates
  - Halt(P,X) = False if P(X) diverges
- Using Halt we can create another application K that does the following:
  - if Halt(X,X) then { while True {} }
- If there is Halt, we can write K
- Note that Halt is total (always answers true or false)
- Now what does K(K) return? If H(K,K) is
  - True → but then K loops forever → Contradiction
  - False → but then K will stop → Contradiction
- Absurd → no Halt can exists!

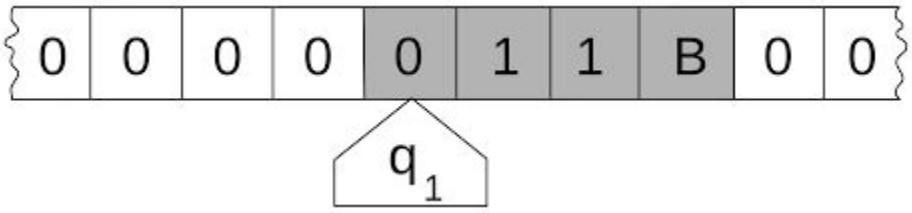
Undecidable problems

- Most interesting properties are undecidable
  - is the function constant?
  - is a given function x equal to function y?
  - does the program always terminate?
  - does the program always diverge?
  - …
- The dream of a computer scientist is destined to remain such...

Expressive power

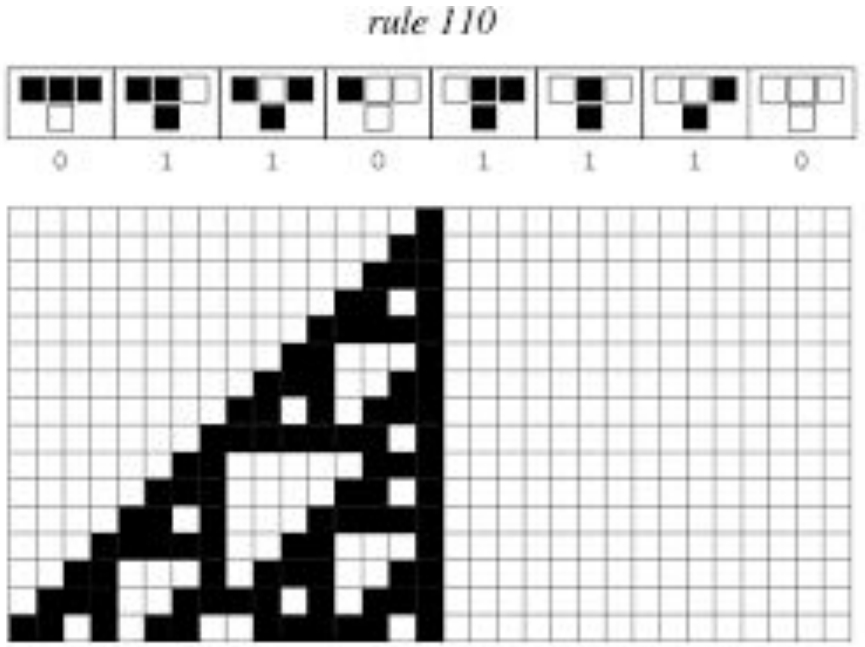
- Obviously there are languages for which the previous properties are decidable...
  - Limit example: A language consisting of the only program

    print(foo)
    - For this language we can decide everything...
    - But we certainly cannot express in it all possible algorithms
- The previous results apply to all formalisms that have the greatest expressive power (i.e., Turing complete)



Other formalism

- Functional programming
- 2 Registry Counter machine (with inc and jump or set to zero instructions)
- Rule 110 cellular automaton



Church Thesis

- Church (~ 1930). A function that can be "computed by any algorithm" coincides with a function calculable by a Turing machine.
  - It is an unproven thesis, since the notion of algorithm is intuitive. However, no counterexample has been seen to date.
- It may be that in the future we discover a more powerful formalism of the Turing Machine (but for now it seems very unlikely)