COMP 102: Computers and Computing
Lecture 17: Computability

Instructor: Kaleem Siddiqi (siddiqi@cim.mcgill.ca)

Class web page: www.cim.mcgill.ca/~siddiqi/102.html

Paris, 1900

- On 8 August 1900, at the Paris 2nd International Congress of Mathematicians, at La Sorbonne.
- German mathematician David Hilbert presented ten problems in mathematics from a list of 23 (1, 2, 6, 7, 8, 13, 16, 19, 21 and 22).
  - The full list was published later.
- The problems were all unsolved at the time, and several of them turned out to be very influential for 20th century mathematics.

Fundamental question

- Can we prove all the mathematical statements that we can formulate ? (Hilbert's 2nd problem)
- Certainly, there are many mathematical problems that we do not know how to solve.
- Is this just because we are not smart enough to find a solution ?
- Or, is there something deeper going on ?

Computer science version of this question

- If my boss / supervisor / teacher formulates a problem to be solved urgently, can I write a program to solve this problem in an efficient manner ???

- Are there some problems that cannot be solved at all ?

- Are there problems that cannot be solved **efficiently** ? (related to Hilbert's 10th problem)

Kurt Gödel

- In 1931, he proved that any formalization of mathematics contains some statements that cannot be proved or disproved.

Alan Turing

- In 1934, he formalized the notion of decidability of a language by a computer.

- What else do we know about Turing?

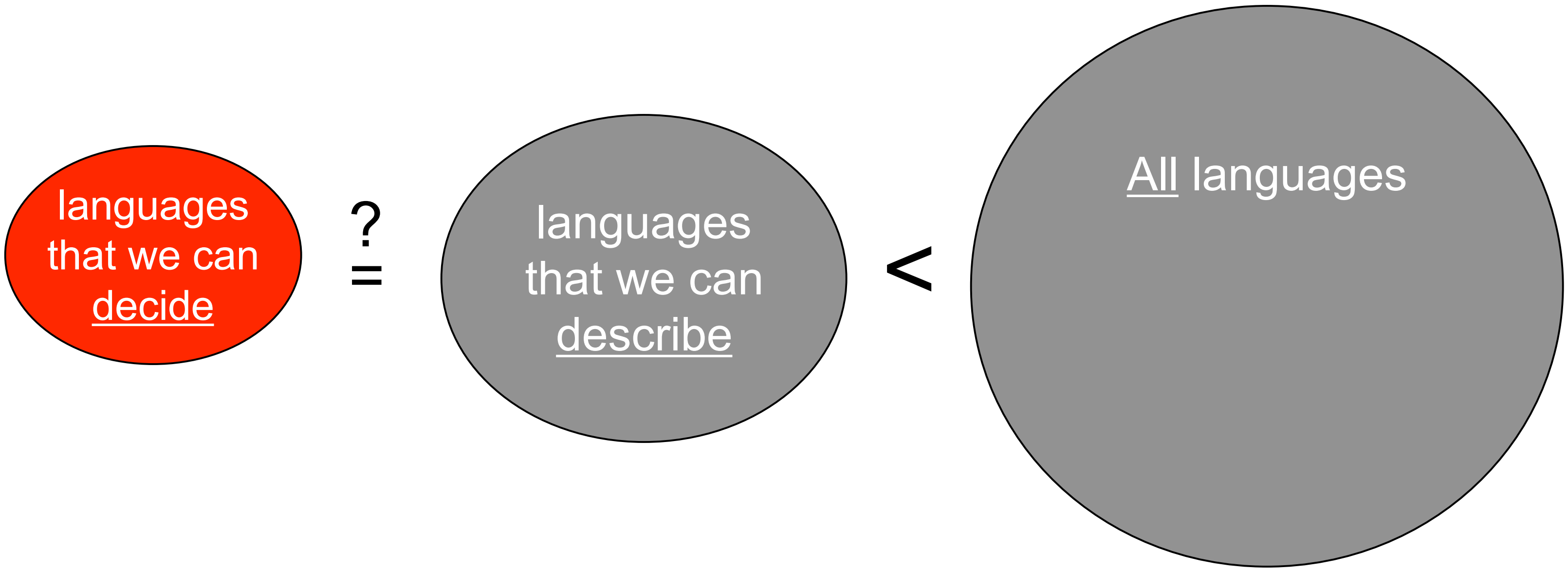
  (Yet more to come…)

A language

- Let Σ be a finite alphabet. (ex: {0,1})

- Let Σ^* be all sequences of elements from this alphabet. (ex: 0, 1, 00000, 0101010101,...)
- A language L is any subset of Σ^*.
- Typically the allowable subsets are specified by the rules of a grammar.
- An algorithm <u>decides</u> a language if it answers Yes when x is in L and No otherwise.

Comparing cardinalities



Alonzo Church

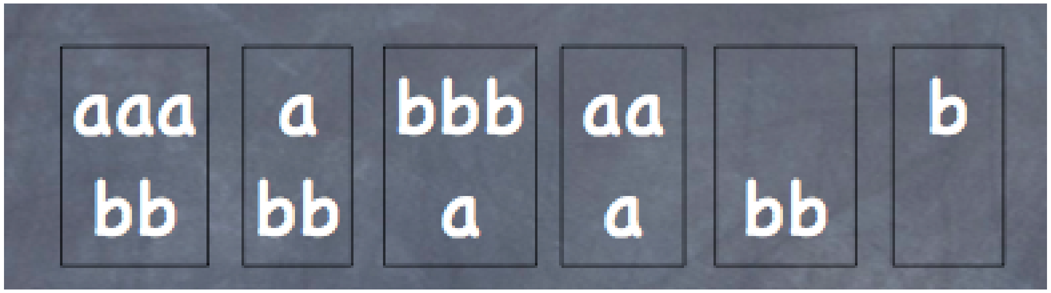
- In 1936, he proved that certain languages cannot be decided by any algorithm whatsoever...

Emil Post

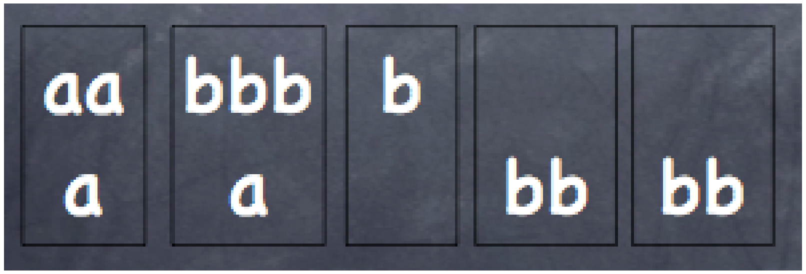
- In 1946, he gave a very natural example of an undecidable language.

Post Correspondence Problem (PCP)

- An instance of PCP with 6 tiles.

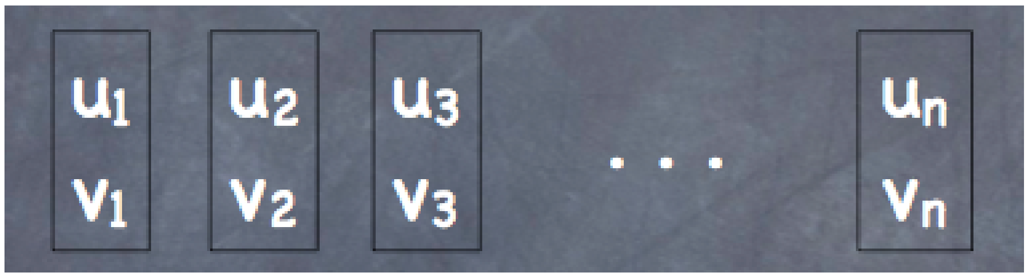


- A solution to PCP.

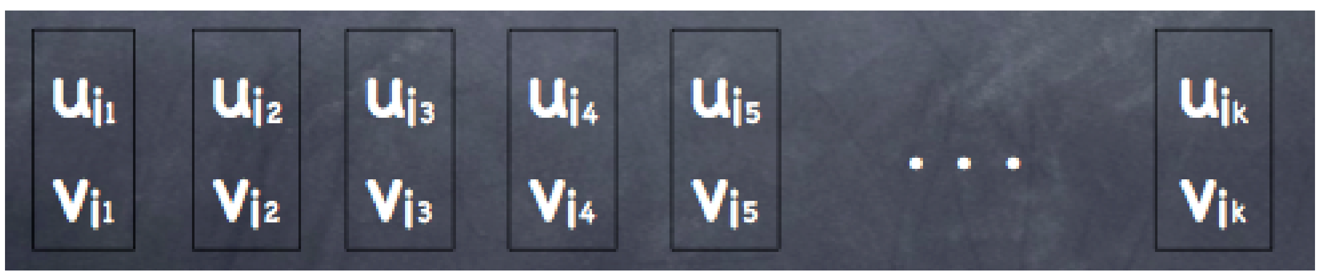


Post Correspondence Problem (PCP)

- Given n tiles, u_1/v_1 ... u_n/v_n where each u_i or v_i is a sequence of letters.



- Is there a k and a sequence $< i_1, i_2, i_3, ..., i_k>$ (with each $1<i_j<n$) such that $u_{i1} \mid u_{i2} \mid u_{i3} \mid ... \mid u_{ik} = v_{i1} \mid v_{i2} \mid v_{i3} \mid ... \mid v_{ik}$?



Post Correspondence Problem (PCP)

- Theorem:

  The Post Correspondence Problem cannot be **decided** by any algorithm (or computer program).

  In particular, **no algorithm can identify in a finite amount of time** the instances that have a **negative outcome**.

  However, if a solution exists, we can find it.

- Proof: Reduction technique - if PCP was decidable, then another problem would be decidable.

The Halting Problem

- Notice that an algorithm is a piece of text.
- An algorithm can receive text as input.
- An algorithm can receive an algorithm as input.

<u>The Halting Problem</u>:

Given two texts A,B, consider A as an algorithm and B as an input.

Will algorithm A halt (as opposed to loop forever) on input B?

The Halting Problem

- Theorem: No algorithm can decide the Halting Problem.

- Proof:

  Assume for a contradiction that an algorithm `Halt(A,B)` exists to decide the Halting Problem. Algorithm A should halt with B as input.

  Consider this algorithm:

```
Bug(A):

If Halt(A,A) then While True do
     { when Halt(A,A) is true then Bug(A) loops }
     { when Halt(A,A) is false then Bug(A) halts }
```

Question: What is the outcome of Bug(Bug)?

The Halting Problem

- If `Bug(Bug)` does not loop forever, it is because `Halt(Bug,Bug)=False`, which means `Bug(Bug)` loops forever.

  Contradiction!

- If `Bug(Bug)` loops forever it is because `Halt(Bug,Bug)=True` which means `Bug(Bug)` does not loop forever.

  Contradiction!

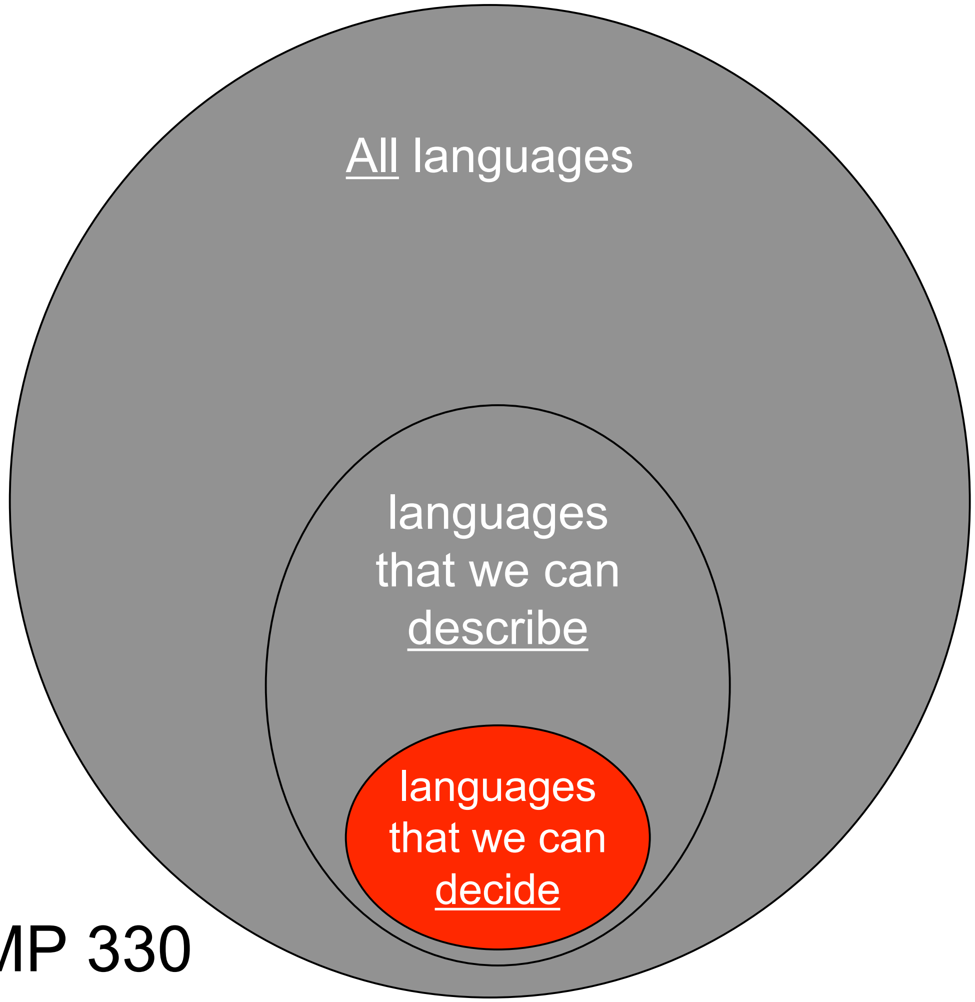
Conclusion: Halt cannot exist.

The Halting Problem and PCP

- Any algorithm to decide PCP can be converted to an algorithm to decide the Halting Problem.
- Also see: http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html

  "Scooping the Loop Snooper"
- Conclusion: PCP cannot be decided either.

Computability Theory



Covered in COMP 330

Decidable Programs

Can we always tell if a program is decidable?

Sometimes we just don't know!

Syracuse Conjecture

For any integer $n>0$, define the following sequence:

$$s_1 = n$$

$$s_{i+1} = \begin{cases} s_i / 2 & \text{if } s_i \text{ is even} \\ 3s_i + 1 & \text{if } s_i \text{ is odd} \end{cases}$$

Then:

$$Syracuse(n) = \begin{cases} \text{least } i \text{ such that } s_1=n, \ldots, s_i=1, \text{ if it exists} \\ 0 \qquad \text{if } s_i \neq 1 \text{ for all } i. \end{cases}$$

Example

- Syracuse(9) = 20

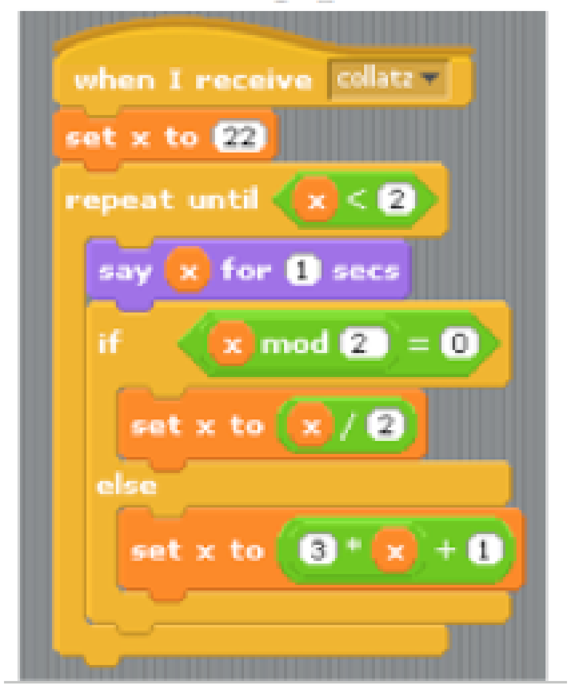
  S_1=9, S_2=28, S_3=14, S_4=7, S_5=22, S_6=11, S_7=34, S_8=17, S_9=52, S_{10}=26, S_{11}=13, S_{12}=40, S_{13}=20, S_{14}=10, S_{15}=5, S_{16}=16, S_{17}=8, S_{18}=4, S_{19}=2, S_{20}=1

- Easy case: Syracuse(2^k) = k+1 for any integer $k \geq 0$

- But not so easy for numbers which are not powers of 2!

Program to calculate Syracuse(n)

- Example for n=22:



Note: "n" is called "x" in this program.

Syracuse Conjecture

- Observation:
  - For all n that we have computed so far, Syracuse(n) > 0.

- Conjecture:
  - For all n>0,   Syracuse(n)>0

But currently, no one knows if this program always stops!

- Problem:
  - If there exists N such that Syracuse(n) = 0, we might not be able to prove it.

Syracuse Conjecture

- The Syracuse conjecture is believed to be true but no proof of that statement was discovered so far.

- It is an **open** problem.

- Even worse, it might be decidable, but there might be no proof that it is decidable !!!

Summary

- There are many problems that turn out to be undecidable.
  - All involve computations that might take an infinite number of operations to solve and you're never quite sure when to stop.

- It is useful to know which programs you should run, and which programs you shouldn't run!

- Showing that a problem is decidable often involves showing that this problem is analogous to another problem which we already know is decidable or not.

  E.g. PCP is not decidable because it is analogous to the Halting Problem.

Take-home message

- Know the difference between:
  - Languages that we can describe.
  - Languages that we can decide.

- Be familiar with the Post Correspondence Problem, and why it is not decidable.
- Understand the general idea of the Halting Problem.
- Be familiar with the Syracuse Conjecture.

Comments

- *http://crypto.CS.McGill.CA/~crepeau/COMP102/*
- *http://www.cs.rutgers.edu/~mlittman/courses/cs105-07b/ch4.pdf*