Computers: What can't they do!
Computers: What can't they do?

Part 1: Decision Problems and Formal Languages

April 9–10, 2012

CS 60: Principles of Computer Science

Let what you say be simply 'Yes' or 'No'; anything more than this comes from evil.

Matthew 5:37 (English Standard Version)

It from bit. Otherwise put, every it—every particle, every field of force, even the space-time continuum itself—derives its function, its meaning, its very existence entirely . . . [from] answers to yes or no questions, binary choices, bits.

John Archibald Wheeler

Theocomp

Computability: What can and can't we compute?

Complexity: How fast can we compute it?

Computability

- ✓ How can I prove that a computer can solve some specific problem?
  - E.g., sorting or the Traveling Salesperson Problem
- ✓ How can I prove that **no** computer can solve a specific problem?
  - If I prove it today, will it still be true tomorrow?

Describing the Problems to be Solved

Assumption 1: Inputs are *finite strings from some finite alphabet of characters.*

- ✓ `"37"`
- ✓ `"3*7=21"`
- ✓ `"sort([3,1,4,1,2], [1,2,3,4])."`

Assumption 2: We care about *decision problems* (i.e., answer is just yes or no)

- ✓ Is 37 prime?
- ✓ Does $3 \times 7 = 21$?
- ✓ Is $[1, 2, 3, 4]$ the result of sorting $[3, 1, 4, 1, 2]$?
- ✓ $\vdots$

Justifying the Assumptions

Assumption 1: Inputs are *finite strings from a finite alphabet of characters*.

- ✓ Generalizes "everything's just a string of bits"

Assumption 2: We care about *decision problems* (i.e., answer is just yes or no)

- ✓ We can solve other problems by asking enough questions.
- ✓ E.g., how could I figure out 3×7 by asking yes/no questions?

Combining the Assumptions

Any *decision problem* is *equivalent to asking*

"Is the input a member of the set L?"

for a suitable set L.

- ✓ For primality testing, $\mathrm{L} = \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$.
- ✓ For multiplication-correctness, $\mathrm{L} = ?$

Conclusion

A one-to-one correspondance:

(computational) problems to solve

sets of finite strings.

These sets are called "formal languages"

Formal Languages: Alphabets

An *alphabet* Σ is a finite, nonempty set.
The elements of Σ are called *letters* or *symbols*.

✓ Examples?

Formal Languages: Strings

A *string over* Σ is a sequence

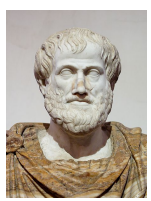
$$c_1 c_2 \cdots c_n$$

where $n \geqslant 0$ and each $c_i \in \Sigma$.

- ✓ Every string is *finite*, but could be arbitrarily long!
- ✓ We will write strings without quotation marks

  xyzzy
- ✓ We write the empty string as ε or λ

  Note: $\varepsilon, \lambda \notin \Sigma$!



What about my alphabet?

Languages

A "*language* L over Σ" is a set of strings over Σ.

- ✓ The set of all strings over Σ is written Σ^*.
- ✓ The empty set $\emptyset$ is a language.
- ✓ The non-empty set $\{\varepsilon\}$ is a language.
- ✓ Languages may be infinite, even if they contain only finite strings!
- ✓ Other examples?

A First Uncomputability Result

- ✓ Every computer program can be represented as a string in $\{0, 1\}^*$.
- ✓ There are as many string/programs as there are natural numbers
- ✓ There are as many languages as there are real numbers.
- ✓ Georg Cantor proved there are *infinitely* many more real numbers than integers!

If you choose a language L "at random," what is the probability there is a program that, for each input, tells you whether that it's in L or not?