The Church-Turing Thesis

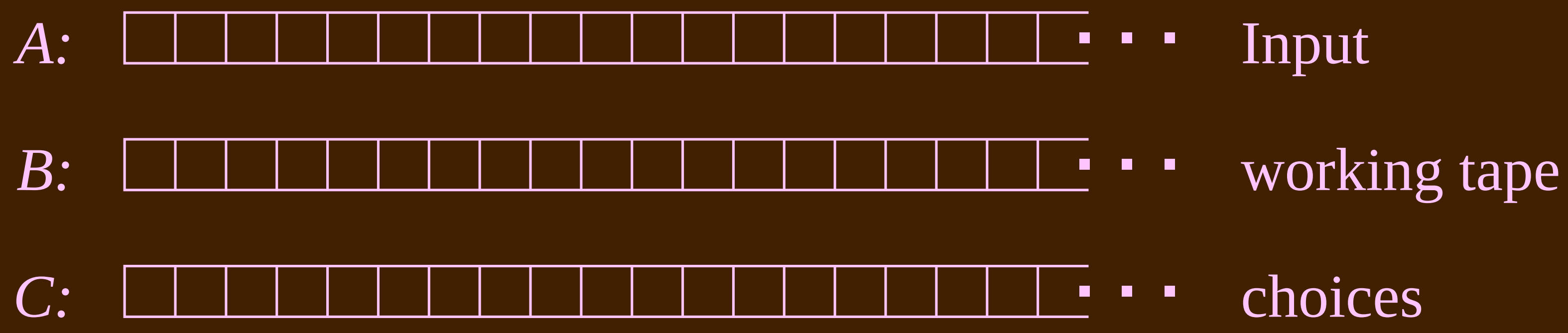
Mark Greenstreet, CpSc 421, Term 1, 2006/07

- Finishing Up Turing Machine Variants
  - Non-Deterministic Turing Machines
  - Addressable memory
- The Church-Turing Thesis
  Anything that can be computed can be computed by a Turing Machine.
- Some Relevant Hilbert Problems
  - Is mathematics complete?
  - Is mathematics consistent?
  - Is mathematics decidable?

Non-Deterministic Turing Machines

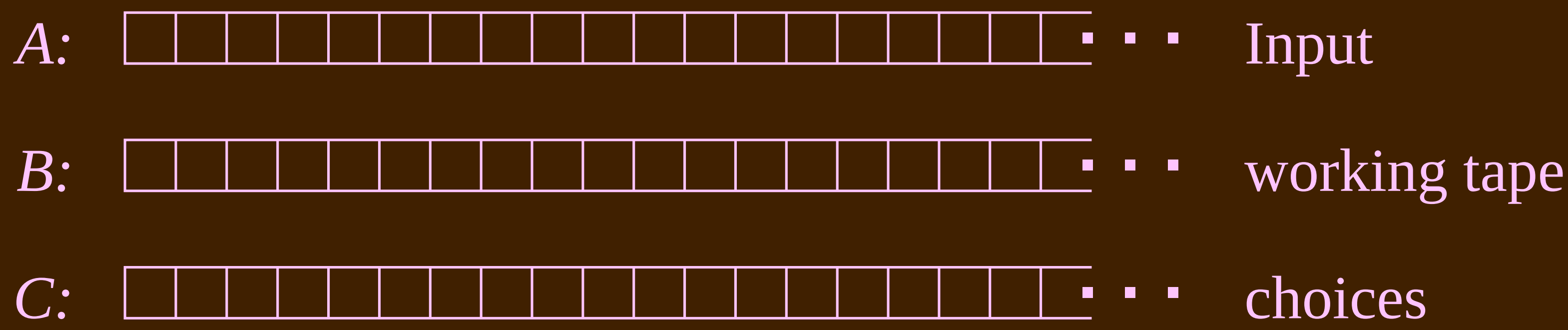
- Like an ordinary Turing machine, but with a transition relation.
  - $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
  - At each step, δ gives the set of possible moves.
  - A non-deterministic TM, N, accepts string w iff there is some set of choices for the moves such that N can reach an accepting state when run with input w.
- Clearly, every deterministic TM is also a non-deterministic TM.
- Can non-deterministic TMs recognize languages that deterministic TMs cannot?

Simulating Non-Determinism



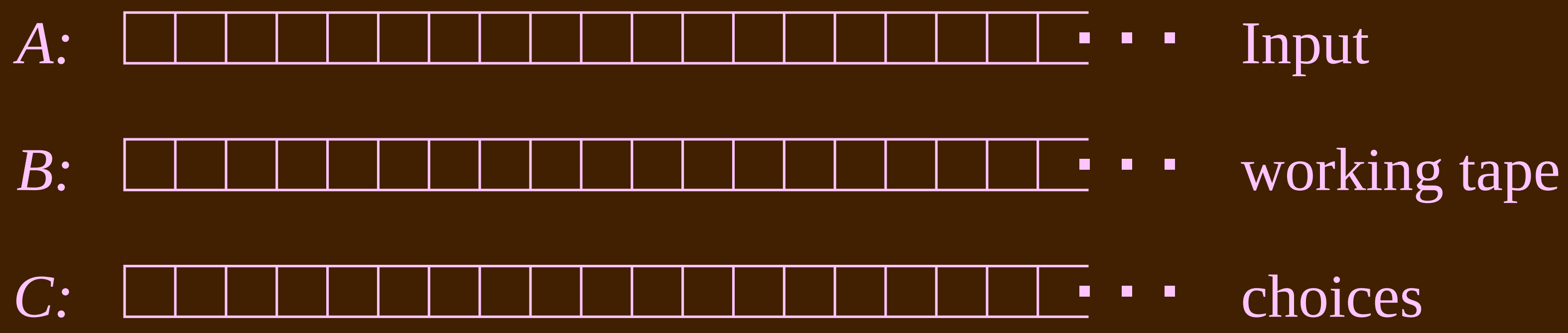
- Three Tapes:
  - A: the input tape. We'll only read from this tape.
  - B: the working tape. We'll simulate the NTM on this tape.
  - C: the choices tape.
    - Let $d = \max_{q,c} |\delta(q, c)|$.
    - The alphabet for C is $\{0 \ldots d - 1\}$.
    - The value of the k^{th} square says what choice to make on the k^{th} move.

Simulating Non-Determinism



- Key Observation
  - If N (the NTM) accept w, it does so after a finite number of moves.
  - Let m be this number.
  - If the m choices are written on tape C, then a DTM can follow the same path as the NTM and accept w.

Simulating Non-Determinism



- The Algorithm:

```
        while(true) {
next:       increment C;
            copy A to B;
            simulate N running on tape B:
                At each step, use C to make the choice;
                If the choice is invalid or blank, go to next;
                If current state of N is reject, go to next;
                If current state of N is accept, accept;
        }
```

Wrapping-Up Non-Determinism

- We've shown that if language L is recognized by an NTM, there is a DTM that recognizes L.
- We haven't shown that if L is decided by an NTM that there is a DTM that decides L – why not?
- In fact, DTMs and NTMs recognize the same class of languages. How can we complete the proof?

Addressable Memory

M: `(0,v0)(1,v1)(2,v2), ···(n,vn)␣␣␣···`

- Add a tape, M, which is a list of (address, value) pairs.
- To read, scan the tape until the addresses match and copy the value to its destination.
- To write, scan the tape until the addresses match and copy the value onto the memory tape, shifting the subsequent values to the right if needed.
- If the address isn't on the tape, create new (address, ϵ) pairs as needed.

More Extensions

- A machine with 32 registers?
- A typical instruction set?
- A stack?
- It seems that we can make a Turing Machine do anything that we associate with a real computer.

The Church-Turing Thesis

- All general purpose computing models:
  - Turing Machines, Java programs, λ-calculus (the basis for lisp and scheme), Gödel's recursive functions, . . .

  are equivalent to each other.
- An algorithm is a finite description of a computation in any of these models.
- The Church-Turing thesis is a conjecture:
  - It's been proven for all of the models above, and for anything anyone has been able to think of.
  - But we can't know for sure that it will apply to anything that comes up in the future.
  - We just know that it's held up extremely well so far, and an exception would seem to need something pretty remarkable.

Hlbert and the Formalist Program

- All of mathematics can be axiomatized (e.g. Peano arithmetic, Zermelo-Fraenkel set theory).
- The notion of a proof can be formalized.
  - If C is a claim, then a proof, P, for C is a sequence of statements in the logic.
  - In these formal systems, checking that P is a valid proof for C can be done completely mechanically, much like a compiler checking a program for syntax or type-checking errors.
- This led Hilbert to propose a grand vision for mathematics.

The Hilbert Questions

- Twenty-three questions that Hilbert raised in a lecture in 1900 as being among the most important questions for mathematicians in the 20^{th} century.
- We'll focus on:
  - Is mathematics complete?
    I.e. Can any true statement be proven?
  - Is mathematics consistent?
    I.e. Is it impossible to prove a contradiction?
  - Is mathematics decidable?
    I.e. Given any claim, is there a procedure by which we can derive a proof for the claim or refute it.
- The last one, like many of Hilbert's questions, asked for a procedure. This goes back to "What is an algorithm?"

What is an Algorithm?

- Prior to Church & Turing: a description of how to compute something.
  - This seems to have been Hilbert's idea in, for example, asking for a procedure with a finite number of steps to determing whether or not a polynomial has an integral root.
  - Gauss and the FFT.
- With Church and Turing, we can be much more precise:
  - We can say what operations are allowed.
  - We can reason about the time and memory required.
  - We can show that there are problems for which no algorithm exists.
- This led to showing the impossibility of solving several of Hilbert's problems, and with it, the impossibility of completing the formalist program.

Reading List:

- Nov. 1: Sipser, 4.1
- Nov. 3: Sipser, 4.2
- Nov. 6: Sipser, 4.2 (cont., midterm 2 cutoff)
- Nov. 8: Sipser, 5.1
- Nov. 10: Sipser, 5.1 (cont.)
- Nov. 13: Remembrance Day (no lecture)
- Nov. 15: Midterm 2
- Nov. 17: Sipser, 5.2
- Nov. 20: Sipser, 5.3