Nondeterministic Turning Machine

- Defined in the expected way
  \[ \delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}) \]
- Definition of Computation:
  - The computation of a nondeterministic Turing machine is a tree whose branches correspond to different possibilities for the machine.
  - If some branch of the computation leads to the accept state, the machine accepts its input.
    - Similar to how nondeterminism is defined in NFA and PDA.
- Let’s not talk about reject yet.
Proof Continued

• Use 3 tapes:
  - input tape (never altered)
  - simulation tape: used to simulate a branch of computation
  - address tape (sequence of choice points we are current trying)

• While
  - Copy input tape to simulation tape
  - Run computation according to choices on address tape
    + Running computation from start to one step after what we previously did
  - If computation goes into an accept state, then accept
  - Otherwise, a choice point is invalid, or we finish the sequence
    + Increment the address tape

For nondeterministic TM $N$, make equivalent deterministic TM $D$

- What do we mean by equivalent?
  + $D$ accepts iff one of the computation branches of $N$ accepts

• Construction
  - Simulate all possible branches of $N$'s nondeterminism
    + at any point, there will only be a finite number of computations
  - Must simulate them all in parallel (breadth-first) rather than depth-first
  - View nondeterminism as a bunch of choice points
    + For each transition, pick an alternative from $\delta(q, a) \subseteq \mathcal{P}(Q \times \Gamma \times \{L, R\})$
  - Let $b$ be the maximum number of choices
    + We can remember a sequence of choices using sequences from $\{1, 2, ..., b\}^*$
      + So a computational path is just a number in base $b$
      + Given a sequence, we can increment it base $b$ to get the next set of choices!
Corollary

**Corollary 3.18**: A language is Turing-recognizable iff some nondeterministic TM recognizes it.

Nondeterminism and Decidability

- Can we define a notion of halting for a nondeterministic TM?
  - Definition: Halts (accepts or rejects) on every computation branch
    + Or computation branch dies out as no transitions from it
  - If any computation branch is still going, we have not halted yet
- A nondeterministic TM is called a **decider** if all branches halt on all inputs
  - In general case, you cannot determine if a NTM is a decider or not
  - But you can still talk about NTMs that are a decider
  - So, if we halted and there is a path that accepts, we accept.
    If we halted and rejected on all paths, we reject
- Aside:
  - What if we accept on one branch, but another branch does not halt?
Equivalence for Deciders

• Does a nondeterministic TM that decides a language have an equivalent deterministic version?

• Previous construction for recognizing a language:

  ```
  While
  Copy input tape to simulation tape
  Run computation according to choices on address tape
  If computation goes into an accept state, then accept
  Otherwise, a choice point is invalid, or we finish the sequence
  Increment the address tape
  ```

• Modify $D$ so it
  + Checks if there are any computation paths still alive for this address length
  + Reject if there are not any
  + If a path accepts, we do not have to wait for all other paths to end
  + We know it is a decider, so it will halt. We don’t have to check this

**Corollary 3.19:** Language is Turing-decidable iff a nondeterministic TM decides it.

Overview

• Nondeterminism
  ⇒ Enumerators

• Algorithms

• Notation
Enumerators

• Type of TM called an enumerator
  - Have a work tape and a printer
    + Starts with blank work tape
  - Can output strings to the printer
    + Can output an infinite list of strings
  - Strings it outputs is its language
    + Don’t worry about repetitions, or order
  - Enumerator TM enumerates the members of its language

• Languages that can be recognized by an Enumerator are called recursively enumerable languages

Recursively Enumerable Languages

• Can we enumerate all numbers, writing them out in base 2?

• Can we enumerate all numbers, writing them out in base \( b \)?

• Can we enumerate all strings in \( \Sigma^* \)?
Equivalence

**Theorem 3.21:** A language is Turing-recognizable iff if some enumerator enumerates it

- Proof $\Leftarrow$ (enumerable language has a TM that recognizes it)
  - We have a an enumerator $E$ that enumerates $L$
  - Construct turing machine $M$ as follows
    $M = "\text{On input } w:\"
    1. Run $E$. Every time that $E$ outputs a string, compare it with $w$
    2. If $w$ ever appears in the output of $E$, accept
  - Clearly $M$ accepts the strings that $E$ outputs and only those strings
  - So, turing machine $M$ recognizes $L$
  - So, $L$ is turing-recognizable

- Proof $\Rightarrow$ (turing-recognizable language can be enumerated)
  - We have a deterministic TM $M$ that recognizes $L$
  - Say $s_1, s_2, s_3...$ is a list of all possible strings in $\Sigma^*$
    - We can make a list of them as $\Sigma^*$ is enumerable
  - Construct $E$ as follows:
    $E = "\text{Ignore the input}\"
    For $i = 1, 2, 3,...$
    For $j = 1, 2, 3,...i$
    Run $M$ for $i$ steps on input $s_j$
    If computation accepts, print out $s_j$
  - If $M$ accepts a particular $w = s_j$, it will do it in say $k$ steps
    - So, $E$ will print it out its outer iteration of $\max(k, j)$ (and many more times)
  - We’re basically running $M$ in parallel on all inputs
Variants

- Pretty well any model with unrestricted access to unlimited memory has the same power
- Similar to how you can
  - Compile one language into another
  - Build an interpreter in one language for another one
- Any two computational models that satisfy certain reasonable requirements can simulate one another and hence are equivalent in power
  - One reasonable requirement is only perform a finite amount of work in a single step
- So, many variants, but still same class of languages
  - Turing-recognizable and Turing-decidable

Overview

- Nondeterminism
- Enumerators
  ⇒ Algorithms
- Notation
Motivation

• Integral roots of a polynomial
  - A polynomial: \(6x^3y^2z^2 + 3xy^2 - x^3 - 10\)
  - A root is an assignment of values to its variables so that the value is 0
  - Integral root is if all variables have an integer value

• Hilbert’s tenth problem was to devise an algorithm that tests whether a polynomial has an integral root
  - Actually, he said ‘a process according to which it can be determined by a finite number of operations’

• As it turns out there is no such algorithm
• But how do we go about proving this? that no algorithm exists
  - What is even an algorithm?

Church-Turing Thesis

• Intuitive notion of algorithm is the same as what can be done with a Turing machine
  - Or with lambda calculus

• Rephrase Hilbert’s tenth problem as a language
  \(D = \{p\mid p\text{ is a polynomial with an integral root}\}\)

• Is \(D\) Turing-recognizable?
  - Let’s first consider polynomials with one variable
    + Try all possible integral values of \(x\): 0,1,-1,2,-2,3,-3,...
    + If polynomial evaluates to 0, accept
  - For multiple variables
    + Can enumerate all possible integral values for the variables
    + If polynomial evaluates to 0, accept

• But is \(D\) decidable?
Overview

- Nondeterminism
- Enumerators
- Algorithms
  ⇒ Notation

Describing TM

- Formal description
  - States and transitions
- Implementation level
  - How the head moves, how it stores data on the tape
- High-level description
  - Describe the algorithm
  - Abstracts away from how machine works. Only use this when comfortable with how TM’s work
Input

• TM’s always take a string as input
• If you want to give it some object, must encode it as a string
  - For object $O_1$, we will refer to its string encoding as $\langle O_1 \rangle$
  - If we want to encode two objects $O_1$ and $O_2$ as input: $\langle O_1, O_2 \rangle$
  - Encode a graph $G$ so a TM can use it as input?
  - Encode a DFA $D$ so a TM can use it as input?
  - Encode a TM $M$ so another TM can use it as input?