The Recursion Theorem

• Lemma: there is a computable function $q: \Sigma^* \rightarrow \Sigma^*$ such that $q(w)$ is a description of a TM $P_w$ that prints out $w$ and then halts.

• Proof:
  - on input $w$, construct TM $P_w$ that has $w$ hard-coded into it; output $<P_w>$

Recall: $q(w)$ is a description of a TM $P_w$ that prints out $w$ and then halts.

Note: $<A> = q(<B>)$
The Recursion Theorem

**Theorem**: Let \( T \) be a TM that computes \( t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \). There is a TM \( R \) that computes the fn:

\( r: \Sigma^* \rightarrow \Sigma^* \)

defined as \( r(w) = t(w, \langle R \rangle) \).

- This allows "obtain own description" as valid step in TM program
- first modify TM so that it takes an additional input (that is own description); use at will

Proof outline: TM \( R \) has 3 parts

1. **Part A**: output description of BT
   - \( \langle A \rangle = q(\langle BT \rangle) \)
2. **Part B**: prepend description of \( A \)
   - read contents of tape \( \langle BT \rangle \)
   - apply \( q \) to it \( q(\langle BT \rangle) = \langle A \rangle \)
   - prepend to tape \( \langle ABT \rangle \)
3. **Part "T"**: run TM \( T \)
   - 2nd argument on tape is description of \( R \)

Summary

- classes of problems:
  - decidable ("solvable by algorithms")
  - recursively enumerable (RE)
  - co-RE

- counting:
  - not all problems are decidable
  - not all problems are RE

- diagonalization: HALT is undecidable
- reductions: other problems undecidable
  - many examples
  - Rice’s Theorem
- natural problems that are not RE
- **Recursion Theorem**: non-obvious capability of TMs: printing out own description
- **Incompleteness Theorem**
Complexity

• So far we have classified problems by whether they have an algorithm at all.
• In real world, we have limited resources with which to run an algorithm:
  – one resource: time
  – another: storage space
• need to further classify decidable problems according to resources they require

Complexity Theory = study of what is computationally feasible (or tractable) with limited resources:
  – running time
  – storage space
  – number of random bits
  – degree of parallelism
  – rounds of interaction
  – others...

Worst-case analysis

• Always measure resource (e.g. running time) in the following way:
  – as a function of the input length
  – value of the fn. is the maximum quantity of resource used over all inputs of given length
  – called “worst-case analysis”
• “input length” is the length of input string, which might encode another object with a separate notion of size

Time complexity

Definition: the running time (“time complexity”) of a TM M is a function \( f : \mathbb{N} \rightarrow \mathbb{N} \) where \( f(n) \) is the maximum number of steps \( M \) uses on any input of length \( n \).
• “\( M \) runs in time \( f(n) \),” “\( M \) is a \( f(n) \) time TM”

Time complexity

• Example: TM \( M \) deciding \( L = \{0^k1^k : k \geq 0\} \).
  
  On input \( x \):
  
  • scan tape left-to-right, reject if 0 to right of 1
  • repeat while 0’s, 1’s on tape:
    • scan, crossing off one 0, one 1
    • if only 0’s or only 1’s remain, reject;
      if neither 0’s nor 1’s remain, accept
  
  \# steps?
Time complexity

- Measure time complexity using asymptotic notation ("big-oh notation")
  - disregard lower-order terms in running time
  - disregard coefficient on highest order term

- example:
  \[ f(n) = 6n^3 + 2n^2 + 100n + 102781 \]
  - \( f(n) \) is order \( n^3 \)
  - write \( f(n) = O(n^3) \)

Asymptotic notation

**Definition:** given functions \( f, g : \mathbb{N} \rightarrow \mathbb{R}^+ \), we say \( f(n) = O(g(n)) \) if there exist positive integers \( c, n_0 \) such that for all \( n \geq n_0 \)

\[ f(n) \leq cg(n). \]

- meaning: \( f(n) \) is (asymptotically) less than or equal to \( g(n) \)
- if \( g > 0 \) can assume \( n_0 = 0 \), by setting \( c' = \max_{0 \leq n \leq n_0} \{ c, f(n)/g(n) \} \)

Asymptotic notation facts

- "logarithmic": \( O(\log n) \)
  - \( \log_b n = (\log_2 n)/(\log_2 b) \)
  - so \( \log_b n = O(\log n) \) for any constant \( b \); therefore suppress base when write it

- "polynomial": \( O(n^k) = n^{O(1)} \)
  - also: \( c^{O(\log n)} = O(n^{\log n}) = n^{O(1)} \)

- "exponential": \( O(2^{n^\delta}) \) for \( \delta > 0 \)

Time complexity

- Recall:
  - language is a set of strings
  - a complexity class is a set of languages
  - complexity classes we’ve seen:
    - Regular Languages, Context-Free Languages, Decidable Languages, RE Languages, co-RE languages

**Definition:** \( \text{TIME}(t(n)) = \{ L : \text{there exists a TM M that decides L in time } O(t(n)) \} \)

- We saw that \( L = \{ 0^k1^k : k \geq 0 \} \) is in \( \text{TIME}(n^2) \).
- Book: it is also in \( \text{TIME}(n \log n) \) by giving a more clever algorithm
- Can prove: There does not exist a (single tape) TM which decides \( L \) in time (asymptotically) less than \( n \log n \)
- How about on a multitape TM?
Time complexity

- 2-tape TM M deciding \( L = \{0^k1^k \mid k \geq 0 \} \).

On input \( x \):
- scan tape left-to-right, reject if 0 to right of 1
- scan 0's on tape 1, copying them to tape 2
- scan 1's on tape 1, crossing off 0's on tape 2
- if all 0's crossed off before done with 1's reject
- if 0's remain after done with ones, reject; otherwise accept.

\[ O(n) + O(n) + O(n) = 3O(n) = O(n) \]

Multitape TMs

- Convenient to "program" multitape TMs rather than single ones
  - equivalent when talking about decidability
  - not equivalent when talking about time complexity

**Theorem:** Let \( t(n) \) satisfy \( t(n) \geq n \). Every multitape TM running in time \( t(n) \) has an equivalent TM running in time \( O(t(n)^3) \).

Multitape TMs

Simulation of \( k \)-tape TM by single-tape TM:

- add new symbol \( x \) for each old \( x \)
- marks location of "virtual heads"

Repeat: \( O(t(n)) \) times

- scan tape, remembering the symbols under each virtual head in the state
- make changes to reflect 1 step of \( M \); if hit #, shift to right to make room.
- when \( M \) halts, erase all but 1st string

\[ O(t(n)) \cdot O(k t(n)) = O(t(n)) \]

Multitape TMs

- Moral: feel free to use \( k \)-tape TMs, but be aware of slowdown in conversion to TM
  - note: if \( t(n) = O(n^c) \) then \( t(n)^2 = O(n^{2c}) \) or \( O(n^c) \)
  - note: if \( t(n) = O(2^n) \) for \( \delta > 0 \) then \( t(n)^2 = O(2^{2n}) = O(2^{\delta n}) \) for \( \delta > 0 \)
- high-level operations you are used to using can be simulated by TM with only polynomial slowdown
  - e.g., copying, moving, incrementing/decrementing, arithmetic operations +, -, *, /

Extended Church-Turing Thesis

- the belief that TMs formalize our intuitive notion of an efficient algorithm is:
  - The "extended" Church-Turing Thesis
    - everything we can compute in time \( t(n) \) on a physical computer can be computed on a Turing Machine in time \( t(n)^3(\text{poly}) \) (polynomial slowdown)
    - quantum computers challenge this belief
Time Complexity

• interested in a coarse classification of problems. For this purpose,
  – treat any polynomial running time as "efficient" or "tractable"
  – treat any exponential running time as inefficient or "intractable"

Key definition: "P" or "polynomial-time" is

\[ P = \bigcup_{k \geq 1} \text{TIME}(n^k) \]