Computability

- regular, context-free and beyond
- \( L = \{1^n | n \text{ and } n + 2 \text{ are prime}\} \)

Alan Turing

- Alan Turing
  - 1912-1954, UK
  - Cambridge University, Princeton University
  - 1936, universal machines and the halting problem
- Alonzo Church
  - 1903-1995, USA
  - Princeton University
  - 1936, lambda calculus and undecidable problems
Turing machines

- The universal machine, a model for computation
- Read, write, move to a new cell, change state
- Tape contains the input string
  All the rest of the tape cells are blank
- Two halting states, Accept and Reject

\[ \begin{array}{cccccccc}
$ & 0 & 1 & 0 & 1 & 0 & 1 & \cdots \\
\end{array} \]

one-way infinite tape

finite state control

University of Sydney
COMP3310/3610: Theory of Computation

Decidable problems

- (definition 3.5p142) A language is called Turing recognizable if some Turing machine recognizes it
  - A recognizer accepts all yes-inputs, may loop for ever on no-inputs
  - Also called recursively enumerable

- (definition 3.6p142) A language is called Turing decidable if some Turing machine decides it
  - always gives an answer
  - Also called recursive
Example

• (example 3.7p143)
  Example of a Turing machine for \(\{0^n | n \geq 0\}\)
  from left to right, cross off every other 0

Multitape Turing machines

• \(k\)-tape Turing machine
  \(\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k\)

• (theorem 3.13p149)
  A \(k\)-tape Turing machine is equivalent to a single tape Turing machine
  Proof: Simulate a multitape machine by a single tape one

Non-deterministic Turing machines

• non-determinism allows several choices for each step of the computation

• \(\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})\)

• many possible next steps

• non-deterministic computation represented as a tree

• Several computation paths

• Computation is accepting if an 'accept' leaf exists

• A non-deterministic branch may be infinite
Non-deterministic Turing machines

- (theorem 3.16p150)
  Every non-deterministic Turing machine has an equivalent
deterministic Turing machine
- Simulate the machine deterministically
- Do all possible computation paths, accept if any one of them
  accepts

We can simulate a given non-deterministic machine $N$ by a 3
tape deterministic machine
- tape 3 is used as an index tape for the non-deterministic
  choices, pointing to a path into the computation tree
- a number '135' means 'take the first choice at the root, then
  the third etc'

- (theorem 3.16p150)
  Every non-deterministic Turing machine has an equivalent
deterministic Turing machine
- Simulate the machine deterministically
- Do all possible computation paths, accept if any one of them
  accepts
- problem: Non-deterministic computation may contain infinite
  paths
  simulate in parallel (breadth first traversal of the computation
  tree)
- Dove-tailing

- tape 1 has the input, tapes 2 and 3 are empty
- copy the input to tape 2
- use tape 2 to simulate the machine $N$. Use the symbols on
  tape 3 for the choices
  - if the symbol describes an unavailable choice goto 4
  - if no more symbols on tape 3, goto 4
  - if rejecting configuration is encountered, goto 4
  - if accept is encountered then {accept}
- replace tape 3 with the next (lexicographically) string
goto 2
Enumerators

- An enumerator is a Turing machine that prints (enumerates) all strings in a language
- (Theorem 3.21p153) a language is Turing recognizable if and only if it has an enumerator

Defining Computation

- Hilbert's tenth problem (1900)
  - Find an algorithm that decides whether a given polynomial has an integral root
  - .."a process by which it can be determined in a finite number of steps"..
- Matijasevic 1970: Hilbert's tenth is unsolvable
  - Martin Davis, Hilary Putnam, Julia Robinson

Church-Turing thesis

- Church-Turing thesis: computable means computable by a Turing machine
- All proposed computation models are equivalent, including
  - Turing machines
  - \( \lambda \)-calculus
  - Recursive functions
  - Quantum Turing machines

Decidable problems for regular languages

- (Theorems 4.1-4.5p166-169)
  - The following problems on regular languages are decidable
    - Does a given DFA \( M \) accept input \( w \)?
    - Same for non-deterministic automata
    - Given a regular expression, does it generate a given string \( w \)?
    - Given a DFA, does it accept any string?
    - Given two DFAs are they equivalent?
Decidable problems for context free

- (theorems 4.7-4.9p170-172)
  the following problems on context free languages are decidable
  - Given a CFG $G$ and a string $w$, does $G$ generate $w$?
  - Given a CFG $G$ is $L(G) = \emptyset$?
  - Every context free language is decidable
- Equivalence is not decidable

Undecidable problems

- A specific problem that is algorithmically unsolvable
- Fundamental limitation of computing
- Example of an unsolvable problem: software verification- Given a program, does it conform to its specification?
- A more simple unsolvable problem: Given a TM $M$ and a string $w$, does $M$ halt on input $w$?
- In fact, (almost) "all" languages are undecidable
- ... and they are not even recognizable
- Existence of undecidable problems

Counting infinities

- How many languages $L \subseteq \Sigma^*$ are there?
  $\Sigma = \{0, 1\}$
- For $L \subseteq \Sigma^k$ there are $2^{2k}$, a finite number
- For $L \subseteq \Sigma^*$ the number of languages is infinite...
- Small infinite sets and big infinite sets
- Georg Cantor, 1873
- Two sets have the same size if the elements of one set can be paired with the elements of the other

Mappings and functions

- A function $f : A \rightarrow B$ is a mapping of the elements of $A$ to the elements of $B$
- $f$ is ...
  - one-to-one if $f(a) \neq f(b)$ whenever $a \neq b$
  - onto if $f(A) = B$
  - correspondence if it is both 1-to-1 and onto
- Two sets $A$ and $B$ have the same size if there is a correspondence $f : A \rightarrow B$ between them.
Countable and uncountable

- The set of natural numbers is $\mathbb{N} = \{0, 1, 2, \ldots \}$
- A set is called **countable** if it is finite or it has the same size with $\mathbb{N}$
- Example: $\mathbb{Q} = \{\frac{m}{n} | m, n \in \mathbb{N}\}$, the set of rational numbers is countable
- A set is called **uncountable** if it is not countable
- The set of real numbers $\mathbb{R}$ is uncountable
- Proofs: by **diagonalization**

Infinite binary strings

- Let $B = \{\text{all infinite binary strings}\}$
- $B$ is uncountable

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 1 & 0 & \ldots \\
1 & 0 & 0 & 1 & 0 & 0 & \ldots \\
2 & 1 & 1 & 0 & 1 & 0 & \ldots \\
3 & 0 & 1 & 1 & 1 & 0 & \ldots \\
4 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

Counting languages and TMs

- **(theorem 4.18p178)** Some languages are not Turing-recognizable
- proof by diagonalization
- the set of all TMs is countable
  - a TM can be described by a finite string over a finite alphabet
  - arrange the strings alphabetically

- the set of all languages over an alphabet $\Sigma$ is uncountable
- $\mathcal{P}(\Sigma^*)$
  - The set of all infinite binary sequences $B$ is uncountable
  - $\Sigma^*$ is countable (arrange alphabetically)
  - Every language corresponds to a characteristic sequence from $B$
More on counting

- Examples of countable sets: correspondence between \( \mathbb{N} \) and \( \mathbb{Z}, \mathbb{N}^2, \{a\}^*, \{a, b\}^* \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>+1</td>
<td>-1</td>
<td>+2</td>
<td>-2</td>
<td>+3</td>
<td>-3</td>
</tr>
<tr>
<td></td>
<td>(0,0)</td>
<td>(0,1)</td>
<td>(1,0)</td>
<td>(0,2)</td>
<td>(1,1)</td>
<td>(2,0)</td>
<td>(0,3)</td>
</tr>
<tr>
<td>( \varepsilon )</td>
<td>a</td>
<td>aa</td>
<td>aaa</td>
<td>aaaa</td>
<td>aaaaa</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>( \varepsilon )</td>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>ab</td>
<td>ba</td>
<td>bb</td>
<td></td>
</tr>
</tbody>
</table>

Counting \( \mathbb{N}^* \)

- Consider \( \mathbb{N}^* \), the set of finite sequences of numbers
- \( \mathbb{N}^* \) is countable, there is a bijection between \( \mathbb{N} \) and \( \mathbb{N}^* \)
- Use the unique prime factorization theorem
  - Every number can be written as a product of prime factors
  - Primes \( p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, \ldots \)
  - Every number can be written as a product of prime factors
    - \( x = p_1^{e_1} p_2^{e_2} \ldots \)
    - \( 525 = p_2 * p_3 * p_3 * p_4 \)

Universal Turing Machine

- Universal Turing Machine \( U \) is given a description of any TM \( M \) and an input \( w \) and simulates \( M \) on \( w \)
- Use 2 tapes. The first tape contains the description of \( M \) and the input \( w \)
  - Write down starting \( <q_0w> \) configuration on the 2nd tape
- Repeat until halting configuration is reached:
  - Replace the configuration on tape 2, according to the transition function of \( M \)
- Accept if \( q_{accept} \) is reached, reject if \( q_{reject} \) is reached
The halting problem

- The halting problem for TMs is undecidable
  \( H_{TM} = \{ < M, w > | M \text{ halts on input } w \} \)
- proof by contradiction
- Assume that \( H_{TM} \) is decidable and \( \text{Halts}(M, w) \) is the TM that decides it
- Define the following TM:
  \[ D(M) := \begin{cases} \text{accept} & \text{if } \text{Halts}(M, M) \text{ then loop for ever} \\ \text{reject} & \text{otherwise halt (accept)} \end{cases} \]

- What is the output of \( D(D) \)? Does it halt?

More on the halting problem

- (theorem 4.11p174,179)
  The acceptance problem for TMs,
  \( A_{TM} = \{ < M, w > | M \text{ accepts input } w \} \) is undecidable
- Proof by contradiction
- Assume \( A_{TM} \) is decidable and \( H(M, w) \) is the TM that decides it

  \[ H(M, w) = \begin{cases} \text{accept} & \text{if } M \text{ accepts input } w \\ \text{reject} & \text{otherwise} \end{cases} \]
More on the halting problem

|    | $M_1$ | $M_2$ | $M_3$ | $M_4$ | ... | $D$ | ...
|----|-------|-------|-------|-------|-----|-----|-----
| $M_1$ | accept | accept | reject | accept | ... | ... | ...
| $M_2$ | reject | accept | accept | reject | ... | ... | ...
| $M_3$ | reject | reject | reject | reject | ... | ... | ...
| $M_4$ | reject | accept | accept | accept | ... | ... | ...
| ... | ... | ... | ... | ... | ... | ... | ...
| $D$ | reject | reject | accept | reject | ... | ... | ...

Un-recognizable languages

- $A_{TM}$ is Turing recognizable
- (theorem 4.22p181)
  A language $L$ is decidable if and only if $L$ and its complement $\overline{L}$ are both Turing recognizable
  - $L$ is called co-Turing-recognizable if its complement $\overline{L}$ is Turing-recognizable
- (corollary 4.23p182)
  $\overline{H}_{TM}$ and $A_{TM}$, the complements of the halting problem and the acceptance problem, are not Turing recognizable

Un-recognizable languages

- $A_{TM}$ is not Turing-recognizable but it is co-Turing-recognizable
- $E_{TM} = \{ G \mid G$ is a TM and $L(G) = \emptyset \}$ is not Turing recognizable
- $E_{Q_{TM}} = \{ < G, H > \mid$ TMs $G, H$ and $L(G) = L(H) \}$ is not even co-TM-recognizable

Classifying problems

Recognizable (RE)

- Decidable (Recursive)
  - Context free
  - Regular
  - co-Recognizable (co-RE)
Classifying problems

- Recognizable (RE)
- Decidable (Recursive)
- Context Free
- Regular

\[
\begin{array}{c}
\text{EQ}_{TM} \\
\text{ATM} \\
\text{co-\text{RE}} \\
(0+1)^* 1 \\
\text{ww} R
\end{array}
\]

Emptiness

- (Theorem 5.2p189) The emptiness problem for TMs is undecidable
- \( E_{TM} = \{ < M > | L(M) = \emptyset \} \)
- Proof: reduce from \( A_{TM} \)
given a TM \( R \) for \( E_{TM} \) build one for \( A_{TM} \)
- Modify \( M \) to accept only \( w \) if it is non-empty
- \( M_1 \): on input \( < x > \)
  - if \( x \neq w \), reject
  - if \( x = w \) simulate \( M \) on \( w \) and accept if \( M(w) \) accepts

Reductions

- Proving undecidability using reductions
- To prove \( L \) is undecidable, prove the following:
  - “if \( L \) is decidable then \( A_{TM} \) is decidable”
- Proofs by contradiction based on a construction
- Given a TM for \( L \) show how you can use it to solve \( A_{TM} \) or any other undecidable problem

Mapping reducibility

- Many different kinds of reductions
- Mapping or many-one reducibility
- (Definition 5.17p206) a function \( f : \Sigma^* \rightarrow \Sigma^* \) is a computable function if some TM on input \( w \) halts with \( f(w) \) on its tape
- (Definition 5.20p207) \( A \) is mapping reducible to \( B \) written \( A \leq_m B \) if there is a computable function \( f : \Sigma^* \rightarrow \Sigma^* \) such that for all \( w \)
  \[
  w \in A \iff f(w) \in B
  \]
The function \( f \) is called the reduction from \( A \) to \( B \)
**Reducibility**

- if $A \leq_m B$ and $B$ is decidable then $A$ is decidable
- if $A \leq_m B$ and $A$ is undecidable then $B$ is undecidable $\leq_m B$
- if $A \leq_m B$ and $B$ is Turing recognizable then $A$ is T- recognizable
- if $A \leq_m B$ and $A$ is not T-recognizable then $B$ is not T- recognizable
- $A \leq_m B$ is the same as $\overline{A} \leq_m \overline{B}$

**Properties of regular languages**

- deciding properties of sets of languages can be a very difficult problem
- automata are simple enough to allow simple algorithms

**Problems on FAs**

- acceptance (or membership): does $M$ accept string $w$? $A_{FA} = \{< M, w > | M \text{ is a FA that accepts } w \}$
- membership for regular expressions
- emptiness: given $A$, is $L(A) = \emptyset$? $E_{FA} = \{< A > | L(A) = \emptyset \}$
- equivalence: do $A$ and $B$ accept the same language? $EQ_{FA} = \{< A, B > | L(A) = L(B) \}$

**Descriptions**

- $< M >$ denotes a description of the machine $M$
- checking if a given string is a valid DFA
- encode everything as a string
- description of a machine and enumeration
- self-reference
Membership

- theorem 4.1p166
- acceptance (or membership): does $M$ accept string $w$? 
  $A_{FA} = \{< M, w > \mid M$ is a FA that accepts $w\}$
- note: this problem deals with all possible DFAs, not a specific instance
- check the description of $M$
- simulate $M$ on $w$

Emptiness

- theorem 4.4p168
- emptiness: given $A$, is $L(A) = \emptyset$? 
  $E_{FA} = \{< A > \mid L(A) = \emptyset\}$
- let $M = (Q, \Sigma, \delta, q_0, F)$ and $|Q| = k$
- check if any final state is reachable from $q_0$ within $k$ steps
- note: we are using the pumping lemma property
  if $M$ accepts any string, then it must accept a short string

Equivalence

- theorem 4.5p169
- equivalence: do $A$ and $B$ accept the same language? 
  $EQ_{FA} = \{< A, B > \mid L(A) = L(B)\}$
- construct the symmetric difference 
  $L' = (L(A) \cap \bar{L}(B)) \cup (\bar{L}(A) \cap L(B))$
- note: the symmetric difference is empty iff $L(A) = L(B)$
- check $L'$ for emptiness

Equivalence of states

- given a DFA $M$, two states $p, q$ are called equivalent if for all strings $w$, 
  starting from state $p$ on input $w$ we end up to an accept iff 
  starting from state $q$ on input $w$ we end up to an accept
- if two states are not equivalent, they are called distinguishable
- testing equivalence: table filling method
Equivalence of states - 1

- construct a table as follows

<table>
<thead>
<tr>
<th>q0</th>
<th>q1</th>
<th>...</th>
<th>q_{k-1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- for states \( p, q \) if there exists an input symbol \( a \) such that \( \delta(p, a) = r \neq s = \delta(q, a) \) then \( p, q \) are distinguishable
- if not, mark the appropriate entry of the table

Equivalence of states - example

- apply previous algorithm for deciding equivalence of two automata
- are the two starting states equivalent?
- use the previous algorithm and treat both automata as one
- if their starting states are found to be equivalent, then the automata are equivalent
Minimizing automata

* (see also problem 7.40p299)
* given a DFA $M$ construct a DFA $M'$ with minimal number of states such that $L(M) = L(M')$
* input $<M>$ where $M = (Q, \Sigma, \delta, q_0, A)$
  output $<M'>$ such that $L(M) = L(M')$
* remove useless states (unreachable from $q_0$)
* find all pairs of equivalent states
* partition into subsets of equivalent states
* merge equivalent states into one new state

Minimality

* Minimized DFAs are ... minimal
* assume not: $M$ is the minimized automaton from the previous algorithm, and there exists an even smaller one $N$
* start states of $M$ and $N$ are equivalent since $L(M) = L(N)$
* if $p, q$ are equivalent then their successors on any one input symbol are equivalent

Minimality- 1

* all states of $M$ are distinguishable (same for $N$)
* all states are reachable in both $M, N$
* every state $p$ of $M$ is equivalent to some state $q$ in $N$ for $s$ such that $M : p_0 \xrightarrow{s} p$ then $N : q_0 \xrightarrow{s} q$
* $N$ is smaller, two states of $M$ are equivalent to the same $N$ state
* therefore they are equivalent - contradiction
Multistack machines

- theorem: a PDA with two stacks is as powerful as a TM
- proof: simulate a TM by a two stack machine
  - one stack holds the TM tape contents to the left of the head and the second the contents to the right of the head

Counter machines

- a $k$-counter machine has access to $k$ counters
- the machine can add/subtract one from each counter or test for zero
- transition depends on state, input symbol, and which counters are zero
- equivalent definition: $k$-stack machine that can push a single symbol on its stacks
- what is the power of a counter machine?

Counter machines

- (theorem) a three counter machine can simulate a TM
  - simulate a 2 stack PDA with two counters
  - each counter has a numeric representation of the stack contents
  - the third is used for adjusting the other two counters
- (theorem) a two counter machine can simulate a TM
  - simulate 3 counters with 2
  - use one counter to store an encoding of the three counters/numbers $(i, j, k)$: store $2^i3^j5^k$
  - use the second counter for adjusting the first
  - incrementing $i$ is multiplying the counter by 2 etc