Turing Machines versus DFAs

TM can both write to and read from the tape

The head can move left and right

The input doesn’t have to be read entirely, and the computation can continue further (even, forever) after all input has been read

Accept and Reject take immediate effect

Deciding the language $L = \{ w\#w \mid w \in \{0,1\}^* \}$

1. If there’s no # on the tape, reject.
2. While there is a bit to the left of #,
   Replace the first bit with X, and check if the first bit to the right of the # is identical. (If not, reject.)
   Replace that bit with an X too.
3. If there’s a bit to the right of #, then reject else accept
Definition: A Turing Machine is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\square \subseteq \Gamma$
- $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$

Turing Machine Configurations

```
q_7
1 1 0 1 0 0 0 1 1 0
```

 corresponds to the configuration:

```
11010q_700110
```

Defining Acceptance and Rejection

Let $C_1$ and $C_2$ be configurations of $M$
We say $C_1$ yields $C_2$ if, after running $M$ in $C_1$
for one step, $M$ is then in configuration $C_2$
Suppose $\delta(q_1, b) = (q_2, c, L)$
Then $aaq_1bb$ yields $aq_2acb$
Suppose $\delta(q_1, a) = (q_2, c, R)$
Then $cabq_1a$ yields $cabcq_2a$

Let $w \in \Sigma^*$ and $M$ be a Turing machine $M$ accepts $w$ if there are configs $C_0, C_1, ..., C_k$, s.t.
- $C_0 = q_0w$
- $C_i$ yields $C_{i+1}$ for $i = 0, ..., k-1$, and
- $C_k$ contains the accepting state $q_{\text{accept}}$

A language is called recognizable (recursively enumerable) if some TM recognizes it
A language is called decidable (recursive) if some TM decides it

{ $0^{2^n}$ | $n \geq 0$ }

PSEUDOCODE:

1. Sweep from left to right, cross out every other 0
2. If in stage 1, the tape had only one 0, accept
3. If in stage 1, the tape had an odd number of 0’s, reject
4. Move the head back to the first input symbol.
5. Go to stage 1.

Why does this work?
Idea: Every time we return to stage 1, the number of 0’s on the tape is halved.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.

\[ C = \{ a^i b^j c^k \mid k = i \cdot j, \text{and } i, j, k \geq 1 \} \]

PSEUDOCODE:
1. If the input doesn't match \( a^i b^j c^k \), reject.
2. Move the head back to the leftmost symbol.
3. Cross off an \( a \), scan to the right until \( b \).
   Sweep between \( b \)'s and \( c \)'s, crossing off one of each until all \( b \)'s are crossed off.
   If all \( c \)'s get crossed off while doing this, reject.
4. Uncross all the \( b \)'s.
   If there's another \( a \) left, then repeat stage 3.
   If all \( a \)'s are crossed out, check if all \( c \)'s are crossed off.
   If yes, then accept, else reject.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

FINITE STATE CONTROL

FINITE STATE CONTROL

FINITE STATE CONTROL

Nondeterministic TMs

Theorem: Every nondeterministic Turing machine N can be transformed into a single tape Turing Machine M that recognizes the same language.

Proof Idea (more details in Sipser):

M(w):
For all strings \( C \in \{Q \cup \Gamma \cup \#\}^* \) in lex. order,
Check if \( C = C_0 \# \ldots \# C_k \) where
\( C_0, \ldots, C_k \) is some accepting computation history for N on w. If so, accept.

Similarly, we can encode DFAs and NFAs as bit strings.

So we can define the following languages:

\( A_{DFA} = \{ (B, w) \mid B \text{ is a DFA that accepts string } w \} \)

\( A_{NFA} = \{ (B, w) \mid B \text{ is an NFA that accepts string } w \} \)

\( A_{TM} = \{ (C, w) \mid C \text{ is a TM that accepts string } w \} \)

(Can define \( (x, y) := 0^{\|xy\|} \))

We can encode a TM as a bit string.

\[
0^n 10^m 10^k 10^s 10^t 10^u 1 \ldots
\]

m tape symbols (first k are input symbols)
accept state
blank symbol

\( ( (p, a), (q, b, L) ) = 0^n 10^a 10^b 10 \)

\( ( (p, a), (q, b, R) ) = 0^n 10^a 10^b 11 \)

A_{DFA} = \{ (D, w) \mid D \text{ is a DFA that accepts string } w \}

Theorem: \( A_{DFA} \) is decidable

Proof Idea: Directly simulate D on w!

A_{NFA} = \{ (N, w) \mid N \text{ is an NFA that accepts string } w \}

Theorem: \( A_{NFA} \) is decidable. (Why?)

A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \}

\( A_{TM} \) is recognizable but not decidable
Universality of Turing Machines

Theorem: There is a universal Turing machine \( U \) that can take as input
- the code of an arbitrary TM \( M \)
- an input string \( w \),
such that \( U(M, w) \) accepts iff \( M(w) \) accepts.

This is a fundamental property:
Turing machines can run their own code!

Note that DFAs/NFAs do not have this property. That is, \( A_{\text{DFA}} \) is not regular.

The Church-Turing Thesis

Everyone’s
Intuitive Notion = Turing Machines
of Algorithms

This thesis is tested every time you write
a program that does something!

A TM recognizes a language if it accepts all
and only those strings in the language

A language is called recognizable or
recursively enumerable, (or r.e.,)
if some TM recognizes it

A TM decides a language if it accepts all
strings in the language and rejects all strings
not in the language

A language is called decidable (or recursive)
if some TM decides it

Recall: Given \( L \subseteq \Sigma^* \), define \( \neg L := \Sigma^* - L \)

Theorem: \( L \) is decidable
iff both \( L \) and \( \neg L \) are recognizable

Given:
a TM \( M_1 \) that recognizes \( A \)
a TM \( M_2 \) that recognizes \( \neg A \),
we can build a new machine \( M \) that decides \( A \)

How? Any ideas?
\( M_1 \) always accepts \( x \), when \( x \) is in \( A \)
\( M_2 \) always accepts \( x \), when \( x \) is not in \( A \)

There are languages over \( \{0,1\} \)
that are not decidable

Assuming the Church-Turing Thesis,
this means there are problems that
NO computing device can solve

We can prove this using a counting argument.
We will show there is no onto function from
the set of all Turing Machines to the set of all
languages over \( \{0,1\} \). (But it works for any \( \Sigma \))

That is, every mapping from Turing machines to
languages must “miss” some language.