A language that is accepted by a Turing machine is called a \textit{recursively enumerable} language.

If a recursively enumerable language is accepted by a Turing machine that halts on all inputs, then the language is called \textit{recursive}.

(it may also be accepted by other Turing machines that do not halt on all inputs)

Of the three examples we saw: \((0+1)^*\), \(0^*\) and \(\{0^n1^n : n \geq 0\}\), the first two are \textit{regular} and the third is \textit{context-free}.

Turing machines can handle more complicated languages.
\{a^n b^n c^n : n > 0\} is a recursive language that is not context-free.

We could extend the ideas from the previous machine but it is easier to introduce multi-tape machines and use one of them.

Multi-track machines

- First understand that the \textit{cells} on the tape of a Turing machine could contain several fields.
- We say the machine has multiple tracks.
- It is not considered to be an extension of the standard machine.

Example

- Two or more tracks are possible:

\[
\begin{array}{cccccccc}
X_1 & X_2 & B & . & . & . & A_r & B & B & B & B \\
B & B & X_3 & . & . & X_r & B & B & B & B \\
\end{array}
\]

where \( n_x(w) \) is the number of occurrences of \( x \) in the string \( w \).
\{ w \in (0+1)^* : n_0(w) = n_1(w) \} \text{ - II}

\text{ATM // required field "A Turin}\\n\text{\{ w in (0 + 1)^* : n0(w)=n1(w) \}}\\n0 1 // \text{input alphabet,}\\n0 1 B // \text{tape alphabet}\\n1 // \text{number of tapes}\\n2 // \text{num. of tracks on tape 0}\\n1 // \text{tape 0 is 1-way infinite}\\nq0 // \text{the initial state}\\nq5 // \text{final state}\\n
\bullet \text{one final state is enough}

\text{CLASS 07- 14} \quad \text{© L. Lander, 2000} \quad \text{CS 573}

\{ w \in (0+1)^* : n_0(w) = n_1(w) \} \text{ - III}

q0 0+B q1 0+0 r // q1: seek 1\\nq0 1+B q2 1+1 r // q2: seek 0\\nq0 0+0 q0 0+0 r\\nq0 1+1 q0 1+1 r\\nq0 B+B q5 B+B r // success\\nq1 0+B q1 0+B r\\nq1 0+0 q1 0+0 r\\nq1 1+1 q1 1+1 r\\nq1 1+B q3 1+1 l // q3: match 1\\nq2 0+B q4 0+0 l // q4: match 0\\nq2 0+0 q2 0+0 r\\nq2 1+1 q2 0+1 r

\text{CLASS 07- 15} \quad \text{© L. Lander, 2000} \quad \text{CS 573}

\{ w \in (0+1)^* : n_0(w) = n_1(w) \} \text{ - IV}

q2 1+B q2 1+B r\\nq3 0+B q3 0+B l\\nq3 1+B q3 1+B l\\nq3 0+0 q0 0+0 r\\nq3 1+1 q3 1+1 l\\nq4 0+B q4 0+B l\\nq4 1+B q4 1+B l\\nq4 0+0 q4 0+0 l\\nq4 1+1 q0 1+1 r\\nend // required

\text{CLASS 07- 16} \quad \text{© L. Lander, 2000} \quad \text{CS 573}

\text{Initial state}

\[\begin{array}{cccccccccccc}
1 & 1 & 0 & 0 & B & B & B & B & B & B & B \\
\end{array}\]

\[\delta(q_0, [1,B]) = (q_2, [1,1], R)\]

\text{CLASS 07- 17} \quad \text{© L. Lander, 2000} \quad \text{CS 573}
have seen a 1, look for a 0

δ(q₂, [1,B]) = (q₂, [1,B], R)

found a 0, move back to the 1 that was marked

δ(q₂, [0,B]) = (q₄, [0,0], L)

move past other 1's

δ(q₄, [1,B]) = (q₄, [1,B], L)

found the marked 1, start a new iteration

δ(q₄, [1,1]) = (q₀, [1,1], R)
back in state \( q_0 \)

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

\[
\delta(q_0, [1,B]) = (q_2, [1,1], R)
\]

a 1 was marked, look for a 0

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

\[
\delta(q_2, [0,0]) = (q_2, [0,0], R)
\]

found a matching 0

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

\[
\delta(q_2, [0,B]) = (q_4, [0,0], L)
\]

go back to the 1 that was marked

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

\[
\delta(q_4, [0,0]) = (q_4, [0,0], L)
\]
found the most recent 1

$$\delta(q_4, [1,1]) = (q_0, [1,1], R)$$

start a new iteration?

$$\delta(q_0, [0,0]) = (q_0, [0,0], R)$$

all the 0's are marked

$$\delta(q_0, [0,0]) = (q_0, [0,0], R)$$

found B+B

$$\delta(q_0, [B,B]) = (q_5, [B, B], R)$$
Now we move up to a new kind of machine: the multi-tape machine.

Single tape machines can simulate multi-tape machines as we shall see later.

A multi-tape Turing machine has several tapes and a read-write head for each tape.

The read-write heads on each tape can move left or right or stay stationary independently.

We can use multi-tape machine for \( \{a^n b^n c^n : n > 0\} \), which is easier to define than a single-tape machine.

We copy the a’s, b’s and c’s onto separate tapes and then compare the lengths.
Back to \( a^n b^n c^n : n > 0 \)

\[ q_0 \rightarrow a + B + B + B \rightarrow q_1 \rightarrow a + d + d + d \rightarrow s + r + r + r \]

Put an end marker on lower tapes

\[ q_1 \rightarrow a + B + B + B \rightarrow q_1 \rightarrow a + a + B + B \rightarrow r + r + s + s \]

a’s copied

\[ q_1 \rightarrow b + B + B + B \rightarrow q_2 \rightarrow b + B + b + B \rightarrow r + s + r + s \]

b’s copied

\[ q_2 \rightarrow c + B + B + B \rightarrow q_3 \rightarrow c + B + B + c \rightarrow r + s + s + r \]
c's copied

\[q_3: B+B+B+B \quad q_4: B+B+B+B\quad s+l+l+l\]

\[\begin{align*}
  a & \quad b & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad a & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad b & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
\end{align*}\]

move back across a's and b's and c's

\[q_4: B+a+b+c \quad q_4: B+a+b+c\quad s+l+l+l\]

\[\begin{align*}
  a & \quad b & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad a & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad b & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
\end{align*}\]

if we reach all the markers, accept

\[q_4: B+d+d+d\quad q_5: B+d+d+d\quad s+r+r+r\]

\[\begin{align*}
  a & \quad b & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad a & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad b & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
\end{align*}\]

the end

\[\begin{align*}
  a & \quad b & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad a & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad b & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
  d & \quad c & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B & \quad B \\
\end{align*}\]
### Format of the file

```
ATM
{a^n b^n c^n : n > 0 }
a b c   // input alphabet, note
a b c d B   // tape alphabet
4   // num. of tapes
1   // num. of tracks on tape 0
1   // num. of tracks on tape 1
1   // num. of tracks on tape 2
1   // num. of tracks on tape 3
1   // tape 0 is 1-way infinite
1   // tape 1 is 1-way infinite
1   // tape 2 is 1-way infinite
1   // tape 3 is 1-way infinite
```

### the transitions

```
q0   // the initial state
q5   // final state—enough
q0 a+B+B+B q1 a+d+d+d s+r+r+r
q1 a+B+B+B q1 a+a+B+B r+r+s+s
q1 b+B+B+B q2 b+B+b+B r+s+r+s
q2 b+B+B+B q2 b+B+b+B r+s+r+s
q2 c+B+B+B q3 c+B+B+c r+s+s+r
q3 c+B+B+B q3 c+B+B+c r+s+s+r
q3 B+B+B+B q4 B+B+B+B s+l+l+l
q4 B+a+b+c q4 B+a+b+c s+l+l+l
q4 B+d+d+d q5 B+d+d+d s+r+r+r
end
```

### Compound states

- It may be easier to form a compound state
- In a compound state we store more visible information, rather than encoding it

### Example (1)

- In our TM for
  \( \{w \in (0+1)^* : n_0(w) = n_1(w)\} \)
  recall the transitions
  
  q0 0+B q1 0+0 r   // q1: seek 1
  q0 1+B q2 1+1 r   // q2: seek 0

- The state \( q1 \) indicates that we have seen an unmatched 0 and we move right to find the corresponding 1
- Similarly \( q2 \) has seen a 1 and we are moving right to a 0
Example (2)

- It is clearer to use compound states:
- let the initial state be \([q_0, B]\)
- the transitions become
  \[
  [q_0, B] \text{ 0B  } [q_1, 0] \text{ 0+0  } \text{r}
  \]
  \[
  [q_0, B] \text{ 1B  } [q_1, 1] \text{ 1+1  } \text{r}
  \]
- The first component \(q_1\) tells we have seen an unmatched symbol and must move right.
- The second component stores which of 0 or 1 was actually seen.

TM File (1)

```
ATM // required field "A Turin
{w in (0 + 1)* : n0(w)=n1(w) }  
0 1 // input alphabet,
0 1 B // tape alphabet
1  // number of tapes
2  // num. of tracks on tape 0
1  // tape 0 is 1-way infinite
[q0,B] // the initial state
[q2,B] // final state
```

TM File (2)

```
[q0,B] 0+B [q1,0] 0+0 r
[q0,B] 1+B [q1,1] 1+1 r
[q0,B] 0+0 [q0,B] 0+0 r
[q0,B] 1+1 [q0,B] 1+1 r
[q0,B] B+B [q2,B] B+B r
[q1,0] 0+B [q1,0] 0+B r
[q1,0] 0+0 [q1,0] 0+0 r
[q1,0] 1+1 [q1,0] 1+1 r
[q1,0] 1+B [q2,0] 1+1 l
[q1,1] 0+B [q2,1] 0+B l
[q1,1] 0+0 [q1,1] 0+0 l
[q1,1] 1+1 [q1,1] 1+1 r
```

TM File (3)

```
[q1,1] 1+B [q1,1] 1+B r
[q2,0] 0+B [q2,0] 0+B l
[q2,0] 0+0 [q0,B] 0+0 r
[q2,0] 1+1 [q2,0] 1+1 l
[q2,1] 1+B [q2,1] 1+B l
[q2,1] 0+0 [q2,1] 0+0 l
[q2,1] 1+1 [q0,B] 1+1 r
end  //required
```

CLASS 07- 46 © L. Lander, 2000 CS 573

CLASS 07- 47 © L. Lander, 2000 CS 573

CLASS 07- 48 © L. Lander, 2000 CS 573

CLASS 07- 49 © L. Lander, 2000 CS 573
Initial state

[q0,B] 1+B [q1,1] 1+1 r

1 1 0 0 B B B B B B B B B B
B B B B B B B B B B B B

have seen a 1, look for a 0

[q1,1] 1+B [q1,1] 1+B r

1 1 0 0 B B B B B B B B B B
1 B B B B B B B B B B B B

found a 0, move back to the 1 that was marked

[q1,1] 0+B [q2,1] 0+B l

1 1 0 0 B B B B B B B B B B
1 B B B B B B B B B B B B

move past other 1’s

[q2,1] 1+B [q2,1] 1+B l

1 1 0 0 B B B B B B B B B B
1 B 0 B B B B B B B B B B

[q2,1]
found the marked 1, start a new iteration

\[ [q2,1] \ 1+1 \ [q0,B] \ 1+1 \ r \]

1 1 0 0 B B B B B B B
1 B 0 B B B B B B B

back in state q0

\[ [q0,B] \ 1+B \ [q1,1] \ 1+1 \ r \]

1 1 0 0 B B B B B B B
1 B 0 B B B B B B B

a 1 was marked, look for a 0

\[ [q1,1] \ 0+0 \ [q1,1] \ 0+0 \ r \]

1 1 0 0 B B B B B B B
1 1 0 B B B B B B B

found a matching 0

\[ [q1,1] \ 0+B \ [q2,1] \ 0+0 \ l \]

1 1 0 0 B B B B B B B
1 1 0 B B B B B B B

1 1 0 B B B B B B B
go back to the 1 that was marked

\[ [q_{2,1}] \ 0+0 \quad [q_{2,1}] \ 0+0 \ 1 \]

```
1 1 0 0 B B B B B B B
1 1 0 0 B B B B B B B
```

found the most recent 1

\[ [q_{2,1}] \ 1+1 \quad [q_{0,B}] \ 1+1 \ r \]

```
1 1 0 0 B B B B B B B
1 1 0 0 B B B B B B B
```

start a new iteration?

\[ [q_{0,B}] \ 0+0 \quad [q_{0,B}] \ 0+0 \ r \]

```
1 1 0 0 B B B B B B B
1 1 0 0 B B B B B B B
```

all the 0's are marked

\[ [q_{0,B}] \ 0+0 \quad [q_{0,B}] \ 0+0 \ r \]

```
1 1 0 0 B B B B B B B
1 1 0 0 B B B B B B B
```
found B+B

[q0,B] B+B [q2,B] B+B r

1 1 0 0 B B B B B B B B B
1 1 0 0 B B B B B B B B B

move to final state and accept

- accept

1 1 0 0 B B B B B B B B B
1 1 0 0 B B B B B B B B B

Special notice

- NOTE THAT IN THE
  TM SIMULATOR
  THERE CAN BE
  NO SPACES
  IN EXPRESSIONS LIKE [q0,1]

Example: shifting over, pick up 1

- Use [q,X,Y] to store the symbols to be shifted:
  [q,B,B] 1 [q,1,B] B r

1 2 4 3 B B B B B B B B B
[q,B,B]
Pick up 2
- \([q,1,B] \ 2 \ [q,2,1] \ B \ r\)

Pick up 4, put down 1
- \([q,2,1] \ 4 \ [q,4,2] \ 1 \ r\)

Pick up 3, put down 2
- \([q,4,2] \ 3 \ [q,3,4] \ 2 \ r\)

Pick up B, put down 4
- \([q,3,4] \ B \ [q,B,3] \ 4 \ r\)
Pick up B, put down 3

- \([q, B, 3] B [q, B, B] 3 r\)

Stop

- If the state is storing blanks and the tape has a blank, stop

\([q, B, B]\)

variable symbols

- The book has the following idea but we cannot use in the current version of the simulator:

  use variables for the tape symbols

could be simple

- Express

  \([q, B, B] 1 [q, 1, B] B r\)
  \([q, 1, B] 2 [q, 2, 1] B r\)
  \([q, 2, 1] 4 [q, 4, 2] 1 r\)
  \([q, 4, 2] 3 [q, 3, 4] 2 r\)
  \([q, 3, 4] B [q, B, 3] 4 r\)
  \([q, B, 3] B [q, B, B] 3 r\)

- as

  \([q, X, Y] Z [q, Z, X] Y r\)

- By the way, my compound states partly reverse the textbook’s
combining machines

- Different TM’s can be combined
- TM’s can be built as a set of “procedures”
- When one “procedure” ends, it calls the next “procedure” to continue
- We can revisit the TM for $\{a^n b^n c^n : n > 0\}$

two machines

- Essentially we had two machines
- Machine 1 marked a “d” on tapes 1, 2 and 3
- Then machine 1 copied the $a$’s from tape 0 onto tape 1, next the $b$’s from tape 0 to tape 2, finally the $c$’s from tape 0 to tape 3

old final = new initial-I

- When that has finished we jump to the initial state (we called it $q4$) of a new TM
- The job of the new TM is to move left past columns of $(a, b, c)$ and to succeed if we arrive at $(d, d, d)$

old final = new initial-II

- You should think of the TM as separate pieces
- Code each piece separately
- Join the pieces together by having the final state of one TM be the initial state of the next TM
Turing Machines

Transition graphs
Encoding Turing Machines

WEB SITE: http://bingweb.binghamton.edu/~lander/cs573.html

A graphical representation of a TM

- We can draw a graph for a TM similar to that for a finite automaton
- The edges are labeled with triples:

\[ q_j \xrightarrow{X/Y/L} q_k \]

- This diagram indicates that
\[ \delta(q_j, X) = (q_k, Y, L) \]

The last example from class 9

- The example of class 7, which had transitions such as
  \[ [q0, B] \rightarrow 1+B \quad [q1, 1] \rightarrow 1+1 \quad r \]
  can be drawn as shown on the next slide
- some unnecessary transitions have been removed
Moving on to Chapter 8

- There are a number of other issues to cover in Chapter 7 but we need to spend time on Chapter 8
- In Chapter 8, we look at a special class of Turing Machines

Restrictions on the TM in Chapter 8

- We assume the input alphabet is always \( \{0, 1\} \)
- We assume the tape alphabet is always \( \{0, 1, B\} \)
- We assume the initial state is always \( q_1, \text{yes ONE} \)
- We assume the unique final state is always \( q_2, \text{yes TWO} \)
- We allow the TM to have other states \( q_3, q_4, \ldots, q_n \text{ but NO } q_0 \)

Only need \( \delta \)

- If we stick to these assumptions, the only thing that changes between Turing Machines is the transition function \( \delta \)
- Once we are given \( \delta \), we can deduce what the reachable states are and we have the whole TM

Gödel

- Now, Gödel had shown that you could encode theorems about arithmetic as (large) numbers
- He then showed that you could encode as a properly encoded theorem something of the following sort:
  “The numeric encoding of this theorem is not provable in the encoding system”
Powerful theories are not complete

- In this way, Gödel showed that given a set of axioms powerful enough to define the natural numbers, it would be possible to state theorems that could not be *proved* true or false.

- Turing looked at similar problems concerning what could be *computed*, also using encoding methods.

### Encoding a TM-I

- Encoding the components of the TM:

<table>
<thead>
<tr>
<th>states</th>
<th>symbols</th>
<th>movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>0$^1$</td>
<td>L 0$^1$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>0$^2$</td>
<td>R 0$^2$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0$^3$</td>
<td>B 0$^3$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_n$</td>
<td>0$^n$</td>
<td></td>
</tr>
</tbody>
</table>

### Encoding a TM-II

- Encoding a transition:

Let $X_1=0$, $X_2=1$, $X_3=B$, $D_1=L$, $D_2=R$

$\delta(q_i, X_j) = (q_k, X_l, D_m)$

- is encoded as

$0^i10^j10^k10^l10^m$

- so a transition is just a binary string

### Encoding a TM-III

- A Turing Machine *[given that we have fixed the alphabets and the initial and final states]* is completely determined by its list of transitions:

000101000100100, 0100100000100010, 0000010001001000100, ...

Encoding a TM-IV

- We might as well put these codes on the same line, separated by “11”

00010100010010011010010000010001
0110000010001000100100010011...

Encoding a TM-V

- We want to look at these encodings as numbers and
- we want to be able to append other binary strings to end of the encodings so
- we surround the whole encoding with “111”

Encoding a TM-VI

- 11100010100010010011010010000010
  001011000001000100100010011...111
- Frightening as it may be, our whole Turing machine is a single positive integer expressed in binary !!!!!

Conversion to a “Chapter 8 TM”

- One result we will have to assume without proof is Theorem 7.10:
- If $L$ is a language over $\{0,1\}$ and $L$ is recognized by some TM, then $L$ is also accepted by a one-tape TM whose tape alphabet is $\{0, 1, B\}$
Examples

- What is the smallest integer that is a true Turing Machine?
  \[111111 = 63\]

- What is the smallest TM with a transition?
  \[1110101010111 = 30039\]

- What is the smallest TM that accepts something (reaches \(q_2\))?
  \[111010100100111 = 119975\] (you have to move right and accept)

A 4-transition machine

- The textbook discusses the example:
  \[
  \begin{align*}
  q_1 & \rightarrow 1 \rightarrow q_3 \rightarrow 0 \rightarrow r \\
  q_3 & \rightarrow 0 \rightarrow q_1 \rightarrow 1 \rightarrow r \\
  q_3 & \rightarrow 1 \rightarrow q_2 \rightarrow 0 \rightarrow r \\
  q_3 & \rightarrow B \rightarrow q_3 \rightarrow 1 \rightarrow l 
  \end{align*}
  \]

- Taking the transitions in this order:
  \[11101001000101001100010101001001100010010010100110001000100010010111 = 268,724,253,279,934,515,351\]

The transitions can be permuted

- Of course, the same TM allows other orders of its transitions:
  \[
  \begin{align*}
  11100001001001010011000100100101001010010100101001010010111 \\
  1110001000100010010110001000100101001011001001010010010111 \\
  111000100010010010010111000100100101001001001010010010111 \\
  111000100010001001011000100100101001001001010010010111 \\
  \end{align*}
  \]

- etc (24 permutations)

Ready to go

- Given the encodings of TM’s, we are almost ready to actually find a language that is not recursively enumerable
- First understand that we know they must exist:
Countable versus uncountable

- \((0 + 1)^*\) is countable but infinite
- Hence, the set of languages over \(\{0, 1\}\) is uncountably infinite
- By Theorem 7.10, every language over \(\{0, 1\}\) that is recognized by a TM is also recognized by one that has a code \(111\ldots 111 = n\)
- There are only countably many positive integers, ...

There are too many languages

- So there are only countably many recursively enumerable languages (languages recognized by TM’s)
- Evidently most languages over \(\{0, 1\}\) are not recursively enumerable
- But can we describe some of those languages?