Recall how we are encoding Chapter 8 TM's

- The textbook discusses the example:
  \[ q_1 \ 1 \ q_3 \ 0 \ r \]
  \[ q_3 \ 0 \ q_1 \ 1 \ r \]
  \[ q_3 \ 1 \ q_2 \ 0 \ r \]
  \[ q_3 \ B \ q_3 \ 1 \ 1 \]

- Taking the transitions in this order:
  \[ 11101001000101001100010101001001 \]
  \[ 0111 \]
  \[ = 268,724,253,279,934,515,351 \]

A non-rec. enum. language

- We write \( w_j \) for the \( j \)-th word and \( M_j \) for the TM whose binary encoding is \( j \) (written in binary)
- \( L_d = \{ w_j : M_j \) does not accept \( w_j \} \)
- How about the complement? \( \overline{L_d} = \{ w_j : M_j \) accepts \( w_j \} \)
- It turns out that this language is recursively enumerable
stating the obvious?

- If $M_j$ accepts $w_j$, we can verify the fact by running $M_j$ on input $w_j$
- However, if $M_j$ does not accept $w_j$, we will never verify this fact by running $M_j$ on input $w_j$ if $M_j$ runs forever on input $w_j$

Universal TM - i

- The more general context for this study is a universal Turing machine
- Our Java program is a sort of Universal TM: it simulates the execution of any other TM on any input string

Universal TM - ii

- We want to build a TM that can simulate any other TM executing on an input string
- We assume the TM that will be simulated is of the Chapter 8 form
- We assume the input string is binary

Universal TM - iii

- First, let $M$ be a Chapter 8-style TM, then any encoding of $M$ as a binary string as described earlier is denoted $\langle M \rangle$
- Usually $\langle M \rangle$ is not unique for a specific $M$
- Let $\langle M, w \rangle$ denote binary encoding of $M$, concatenated with a binary input string $w$
For example, \( w \) might be 1100010101:

\[
11101001000101010001000101010000100110010001010011000100010001001001010011000100010001000100100111111010101
\]

A universal TM takes as input a binary string \(<M, w>\):

- Any universal TM accepts \(<M, w>\) if \(M\) accepts \(w\), otherwise \(<M, w>\) is not accepted.

We build a 3 tape machine:

- Tape 1 is the input containing \(<M, w>\).
- Tape 2 will simulate the tape of \(M\).
- Tape 3 stores the control state of \(M\) in unary encoding.

Start:

- Contains \(<M, w>\).
- Blank but will contain \(w\).
- Blank but will contain 0.

Go through a few states, checking format, copy \(w\) to tape 2, put 0 (= \(q_1\)) on tape 3, the \(c\) is for convenience.

Return to start of tapes.
通用TM - viii

- 重复以下操作：检查磁带3是否包含00，如果包含，则停止并成功。

通用TM - ix

- 如果磁带3包含$0^i$, $i \neq 2$，且磁带2的读头正在读取$X_i$ ($X_1$是0，$X_2$是1，$X_3$是B)，则在磁带1中搜索模式$110^i10^j10^k10^l10^m11$。

通用TM - x

- 模拟确切的移动到磁带2和3：在磁带3上放$k$个0，在磁带2的读头上放$X_l$，然后移动方向$D_m$（$D_1$=L, $D_2$=R)。

The machine halts if $M$ does

- 我们正在模拟$M$，通过读取其磁带1上的过渡，并在磁带2上执行它们，将当前状态（编码）存储在磁带3上。

- 如果$M$在$w$上停止，该模拟器将停止，要么是因为$M$到达$q_2$（由磁带3上的00识别）或因为没有下一个过渡，$M$未能接受（模拟也是如此）。

Class 13 - 18 © L. Lander, 2000 18CS 573

Class 13 - 19 © L. Lander, 2000 19CS 573

Class 13 - 20 © L. Lander, 2000 20CS 573

Class 13 - 21 © L. Lander, 2000 21CS 573
The universal machine can run for ever

- If $M$ runs for ever on input $w$ without reaching $q_2$ then the simulation will also run for ever
- The **UNIVERSAL LANGUAGE** is $L_u = \{<M, w>: M \text{ accepts } w\}$
- A **UNIVERSAL TURING MACHINE**, $M_u$ such as the one just built, is any TM whose language is $L_u$
- Assume $M_u$ is converted to the Chapter 8 format

---

$L_u$ is not recursive

- We use the fact the $L_d$ is not recursively enumerable to prove $L_u$ is not recursive
- We need to prove Theorem 8.1: The complement of a recursive language is also recursive

---

**Proof of Theorem 8.1**

- Suppose we have a TM $M$ that halts on all inputs and recognizes the recursive language $L$. Assume no transitions from final states.
- Build a complementary TM $M'$ to recognize the complement of $L$:
- Make all the final states non-final, add one new state $p$ and make it final, for every undefined transition $\delta(q, X)$, where $q$ was not final, add a transition $\delta(q, X) = (p, X, R)$

---

**Fixing up the left end of the tape**

- Add more transitions to $M'$ so that it initially moves the input one cell to the right and puts a marker “¢” at the left-hand end of the tape
- Add the transition $\delta(q, \) = (p, \, R)$ for all states
- $M'$ starts simulating $M$ with the tape-head on the 2nd cell (start of original input)
- $M$ would have failed if we reach “¢” but $M'$ succeeds in this case
$M'$ is the complementary machine

- The result is a machine that rejects the strings that $M$ accepted.
- Further, when $M$ halted and failed, we have now added a transition to a final state $p$, so $M'$ accepts.
- Similarly, if $M$ fails by coming off the tape at the left end, $M'$ succeeds.
- This $M'$ halts on all inputs and accepts the complement of $L$.

Proof that $L_u$ is not recursive

- We prove that $L_u$ is not recursive by contradiction.
- We show that if $L_u$ is recursive then $\overline{L_d}$ is recursive.
- By Theorem 8.1, it would follow that the complement of $\overline{L_d}$, i.e. $L_d$ is recursive.
- But $L_u$ is not even recursive, so it is certainly not recursive.
- Hence $L_u$ cannot be recursive.

The connection between $L_u$ and $\overline{L_d}$

- The only missing piece is to show that if $L_u$ is recursive then $\overline{L_d}$ is recursive.
- Suppose $M$ is a halting TM for the language $L_u$.
- Create a halting TM for $\overline{L_d}$ as follows:
  - The input to the new TM $M'$ is a string $w$.

Find where $w$ is in canonical order

- On separate tapes generate:
  - the numbers $j = 1, 2, 3, 4, 5, \ldots$ in binary and
  - the strings over $\{0, 1\}$ in canonical order $w_j = \varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$
- each time you generate a new $j$ or $w_j$, replace the previous one and
- check if $w_j$ is equal to the input $w$. 
you already have $j$, so form $jw$ and run $M$

- As soon as you find $w_j = w$ (which must occur), we copy the binary integer $j$ onto a tape followed by $w_j$
- This is exactly the string $<M_j, w_j>$
- We then run $M$ on this input
- $M$ always halts by hypothesis
- If $M$ accepts then $M_j$ accepts $w_j$ and $w = w_j$ is in $\overline{L_d}$
- If $M$ rejects then $M_j$ rejects $w_j$ and $w = w_j$ is not in $\overline{L_d}$

$M$ does not exist

- Actually if we just reversed the accept/reject outputs of $M$ at this point we would get a halting TM for $L_d$
- Since $L_d$ cannot be recursive, nor can $\overline{L_d}$ be recursive. This $M'$ does not exist
- But $M'$ exists if $M$ does, so...
- $M$ does not exist: $L_u$ is not recursive

However $\overline{L_d}$ is rec. enum.

- Instead of working with the fictitious machine $M$, use any universal TM $M_u$ for $L_u$
- Form the machine $M'$, which begins as before to discover the $j$ and $w_j$, such that $w_j = w$
- Then run $M_u$ on $<M_j, w_j>$
- $M_u$ accepts if and only if $M_j$ accepts $w_j$, i.e. $w$ is in $\overline{L_d}$
- $M'$ is a TM for $\overline{L_d}$, which is rec. enum.

we will identify a “recursive” language

- $L_d$
- $\overline{L_u}$
- $\{0^n1^n : n > 0\}$
- $\Sigma^*$
- regular
- $0^*1^*$

Recursively enumerable

Recursive

Context-free

Class 13 - 30 © L. Lander, 2000 30CS 573

Class 13 - 31 © L. Lander, 2000 31CS 573

Class 13 - 32 © L. Lander, 2000 32CS 573

Class 13 - 33 © L. Lander, 2000 33CS 573
Another piece of the puzzle

- We will need the Theorem 8.3:
  *If a language \(L\) is rec. enum. and if the complement \(\overline{L}\) is also rec. enum., then both \(L\) and \(\overline{L}\) are recursive*

Proof of Theorem 8.3

- Suppose \(M_1\) is the TM for \(L\) and \(M_2\) is the TM for \(\overline{L}\) (perhaps neither machine halts on all inputs)
- Create a combined TM that takes input string \(w\) and simulates \(M_1\) and \(M_2\) simultaneously, using one tape for each and keeping track of the states of each separately \([q_1, q_2]\)

- The machine always halts

  tape for \(M_1\) \(X_1\)
  tape for \(M_2\) \(X_2\)

  \([q_1, q_2]\)

- (1) If we reach a final state in the \(q_1\) component, halt and ACCEPT (\(M_1\) accepts: in \(L\))
- (2) If we reach a final state in the \(q_2\) component, halt and REJECT (\(M_2\) accepts: \(w\) is in the complement of \(L\))
- (3) \(M_1\) halts in a non-final (rejecting) state or passes the left end of its tape, then halt and fail (\(M_1\) rejects: \(w\) is in the complement of \(L\))
- (4) \(M_2\) halts in a non-final (rejecting) state or passes the left end of its tape, then halt and accept (\(M_2\) rejects: \(w\) is in the \(L\))
Everything covered

- There are no other possibilities because, even if one machine were to run for ever on w (hence not accept), the other would have to accept (hence halt)

Turing Machines-cont’d

Recursively enumerable languages that are not recursive

Textbook Example 8.2

- Consider two other languages:
  \[ L_e = \{ <M> : L(M) = \emptyset \} \]
  \[ L_{ne} = \{ <M> : L(M) \neq \emptyset \} = L_e \]
- The textbook shows that is \( L_{ne} \) rec. enum. but not recursive and is not \( L_e \) rec. enum.

The intuition

- We can find out if a TM recognizes something (\( L_{ne} \) case) by looking for the string that it accepts, halting as soon as we find it
- However, in \( L_e \) how can we tell a machine does not accept a string? If we just keep testing strings, we will test for ever
Proof for $L_{ne}$

- (I use the deterministic version: it seems more convincing)
- The input is a binary encoding of a Turing machine $<M>$

Setting up for an iteration

- Put $n$ 0’s on Tape 1 (start with two 0’s and keep adding one zero on each iteration)
- For each new $n$, start with one 0 on Tape 2 and $(n - 1)$ 0’s on Tape 3
- Keep the $n$ on Tape 1 fixed through the following steps

One step during the iteration

- Suppose we have reached $j$ 0’s on Tape 2 and $(n - j)$ 0’s on Tape 3
- Generate the $j$-th word $w_j$ in the canonical order
- Simulate $M$ on $w_j$ for $(n - j)$ steps only

When to halt

- If $M$ accepts $w_j$ within $(n - j)$ steps, halt and accept
- If not, increase the number of 0’s on Tape 2 (to $j + 1$) and decrease the 0’s on Tape 3 (to $n - j - 1$)
- When $j$ reaches $n$, add another 0 on Tape 1 (to get $n + 1$) and set Tapes 2 and 3 back to 0 and $0^n$
- Keep iterating until $M$ accepts
We recognize $L_{ne}$

- Obviously, if $M$ accepts a string, our procedure will eventually recognize that, otherwise our procedure fails to halt.
- Why didn’t we just generate the $w_j$ on one tape and simulate $M$ on each one until we find string $M$ accepts?
- Because somewhere along the way we might get a $w_j$ on which $M$ runs for ever...before we reach the string that $M$ does accept.

What about $L_e$?

- To prove that $L_e$ is not rec. enum., we show that if it were, then $L_u$ would be recursive (which it is not).
- Suppose $L_e$ were rec. enum., then $L_e$ and its complement $L_{ne}$ would both be rec. enum.
- By theorem 8.3, that would mean that both $L_e$ and $L_{ne}$ were recursive.

Construct a strange machine

- Suppose we take the typical input for $L_u$, which is a pair $<M, w>$
- Construct a new Turing machine $M'$, using the input $<M, w>$, that has the following property:
  - $L(M') = \emptyset$ if and only if $<M, w> \notin L_u$
  - $L(M') = \{0, 1\}^*$ if and only if $<M, w> \in L_u$

Constructing $M'$

- The transitions are given in the textbook but here is the idea:
- Whatever the input given to $M'$ may be (say the input is $x$), all the TM does is simulate $M$ on $w$
- $M'$ accepts $x$ if $M$ accepts $w$
If $M$ does not accept $w$ or if $M$ runs for ever, then $M'$ does not accept $x$.

Clearly if $M$ accepts $w$, then $M'$ accepts every $x$, i.e. $L(M') = (0+1)^*$.

If $M$ does not accept $w$, then $M'$ accepts nothing, i.e. $L(M') = \emptyset$.

Suppose there is a halting TM $M_e$ (an algorithm) that recognizes $L_e$.

Then we use Theorem 7.10 to convert the TM $M'$ to $M''$, which has the Chapter 8 form.

Convert $M''$ to one of its binary representations $<M''>$.

Feed $<M''>$ to $M_e$.

Get the contradiction.

If $M_e$ accepts $<M''>$ then $L(M'') = \emptyset$ so that $M$ does not accept $w$, i.e. $<M, w>$ is not in $L_u$.

If $M_e$ rejects $<M''>$ then $L(M'') = (0+1)^*$ so that $M$ accepts $w$, i.e. $<M, w>$ is in $L_u$.

Overall we have an algorithm for $L_u$.

We have a contradiction: $L_e$ recursive (since $L_{ne}$ recursive enum.) $\Rightarrow$ $L_u$ recursive (false).
Recall

- Do you remember why $L_u$ is not recursive?
- It is because if it were, then $L_d$ would be recursive
- In that case, $L_d$ would be recursively enumerable, which it is not
- The proof that $L_d$ is not recursively enumerable is the foundation of all the other proofs

Remember that clever paradox?

- We ask if $w_k$ is in $L_d$
  - $w_k$ is in $L_d$
    - if and only if
  - $w_k$ is accepted by any TM for $L_d$
    - if and only if
  - $w_k$ is accepted $M_k$
    - if and only if
  - $w_k$ is not in $L_d$