COURSE: THEORY OF AUTOMATA COMPUTATION

TOPICS TO BE COVERED

- Relationship Between the Classes of Recursively Enumerable and Recursive Languages

RELATIONSHIP BETWEEN RE AND RECURSIVE LANGUAGES

Theorem: If L is a recursive language, then L is recursively enumerable.

Proof:

Let L be a recursive language over Σ.

Then, there is a TM T deciding L.

Then, T also accepts L.

Thus, L is recursively enumerable.

RELATIONSHIP BETWEEN RE AND RECURSIVE LANGUAGES

Theorem: Let L be a language. If L and $\overline{L}$ are recursively enumerable, then L is recursive.

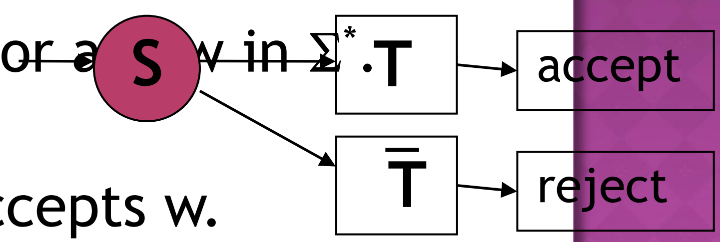
Proof:

Let L and $\overline{L}$ be recursively-enumerable languages over Σ.

Then, there are a TM T accepting L, and a TM $\overline{T}$ accepting $\overline{L}$.

For any string w in Σ^*, w is either in L or in $\overline{L}$.

That is, either T or $\overline{T}$ must halt on w, for any w in Σ^*.



We construct an NTM M as follows:

If w is in L, T halts on w and thus, M accepts w.

If w is not in L, $\overline{T}$ halts on w and thus, M rejects w.

Then, M computes the characteristic function of L. Then, L is recursive.

DECISION PROBLEMS

- A decision problem is a prob. whose ans. is either yes or no
- A yes-instance (or no-instance) of a problem P is the instance of P whose answer is yes (or no, respectively)
- A decision problem P can be encoded by f_e over Σ as a language $\{f_e(X) \mid X$ is a yes-instance of $P\}$.

ENCODING OF DECISION PROBLEMS

- Is X a prime ?

  $\{1^X \mid X \text{ is a prime}\}$
- Does TM T accept string e(T)?

  {e(T) | T is a TM accepting string e(T)}
- Does TM T accept string w?

  {e(T)e(w) | T is a TM accepting string w} or

  {<T,w> | T is a TM accepting string w}

DECIDABLE (OR SOLVABLE) PROBLEMS

Definition:

If f_e is a reasonable encoding of a decision problem P over Σ, we say P is decidable (or solvable) if the associated language $\{f_e(X) \mid X$ is a yes-instance of $P\}$ is recursive.

A problem P is undecidable (or unsolvable) if P is not decidable.

SELF-ACCEPTING

- SA (Self-accepting) = $\{w \in \{0,1,\#,\ ,\}^* \mid w = e(T)$ for some TM T and $w \in L(T)\}$

- NSA (Non-self-accepting) = $\{w \in \{0,1,\#,\ ,\}^* \mid w = e(T)$ for some TM T and $w \notin L(T)\}$

- E (Encoded-TM) = $\{w \in \{0,1,\#,\ ,\}^* \mid w = e(T)$ for some TM T$\}$

NSA IS NOT RECURSIVELY ENUMERABLE

We prove by contradiction.
Assume NSA is recursively enumerable.
Then, there is TM T_0 such that $L(T_0)$=NSA.
Is $e(T_0)$ in NSA?

- If $e(T_0) \in$ NSA, then $e(T_0) \notin L(T_0)$ by the definition of NSA But $L(T_0)$=NSA. Thus, contradiction.
- If $e(T_0) \notin$ NSA, then $e(T_0) \in$ SA and $e(T_0) \in L(T_0)$ by the definition of SA. But $L(T_0)$=NSA. Thus, contradiction.

Then, the assumption is false.
That is, NSA is not recursively enumerable.

E IS RECURSIVE

<u>Theorem:</u> E is recursive.

Proof:

We can construct a regular expression for E according to the definition of the encoding function as follows:

$R = S\ 1\ (M\ \#)^+$

$S = 0$

$M = Q\ ,\ A\ ,\ Q\ ,\ A\ ,\ D$

$Q = 0^+$

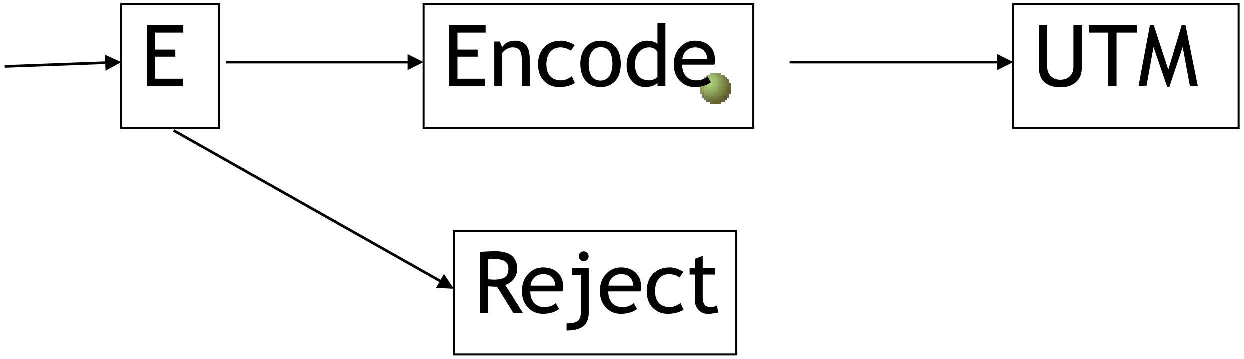
$A = 0^+$

$D = 0 + 00 + 000$

Then, E is regular, and thus recursive.

SA IS RECURSIVELY ENUMERABLE

- Construct a TM S accepting SA
- If w is not e(T) for some TM T, S rejects w.
- If w is e(T) for some TM T, S accepts e(T) iff T accepts e(T).
- L(S) = {w| w=e(T) for some TM T accepting e(T) = SA.
- Then, SA is recursively enumerable.



SA IS NOT RECURSIVE

- NSA = E – SA
- NSA is not recursively enumerable (from previous theorem), and thus not recursive.
- But E is recursive.
- From the closure property, if L1 and L2 are recursive, then L1 - L2 is recursive.
- Using its contrapositive, if L1 - L2 is not recursive, then L1 or L2 are not recursive.
- Since NSA is not recursive and E is recursive, SA is not recursive.

CO-R.E.

Definition

- A language L is co-R.E. if its complement $\overline{L}$ is R.E.
- It does not mean L is not R.E.

Examples:

- SA is R.E. $\overline{SA} = \overline{E} \cup NSA$ is not R.E.
  - $\overline{SA}$ is co-R.E., but not R.E.
- NSA is not R.E. $\overline{NSA} = \overline{E} \cup SA$ is R.E.
  - NSA is co-R.E., but not R.E.
- E is recursive, R.E., and co-R.E.

RELATIONSHIP BETWEEN R.E., CO-R.E. AND RECURSIVE LANGUAGES

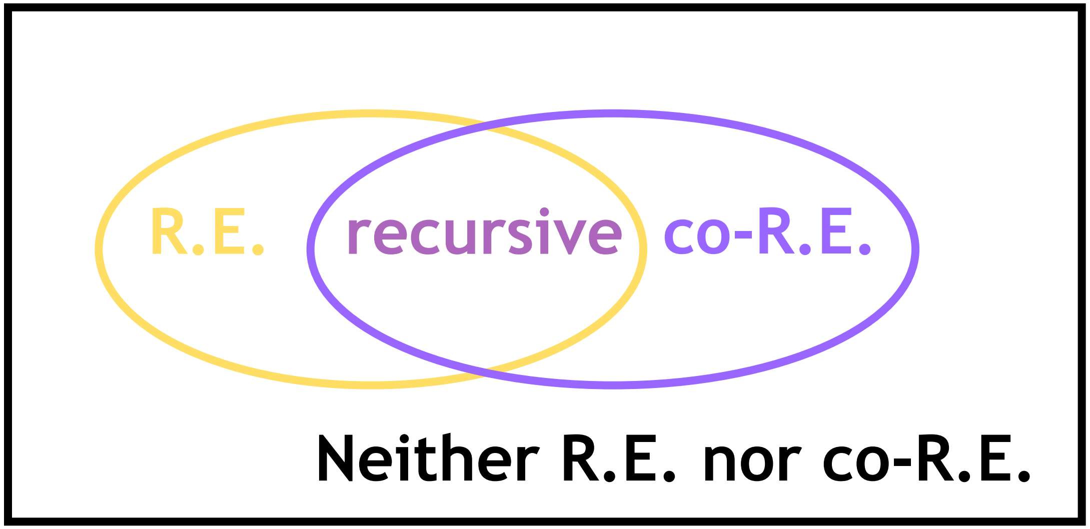
Theorem: Let L be any language. L is R.E. and co-R.E. iff L is recursive.

Proof:

- ($\rightarrow$) Let L be R.E. and co-R.E. Then, $\overline{L}$ is R.E. Thus, L is recursive.
- ($\leftarrow$) Let L be recursive. Then, L is R.E. From the closure under complementation of the class of recursive languages, $\overline{L}$ is also recursive. Then, $\overline{L}$ is also R.E. Thus, L is co-R.E.

OBSERVATION

- A language L is either
  - recursive
  - R.E., bot not recursive
  - co-R.E., but not recursive
  - Neither R.E. nor co-R.E.



REDUCTION

Definition:

Let L_1 and L_2 be languages over Σ_1 and Σ_2, respectively. L_1 is (many-one) reducible to L_2, denoted by $L_1 \leq L_2$, if there is a TM M computing a function f: $\Sigma_1^* \rightarrow \Sigma_2^*$ such that $w \in L_1 \leftrightarrow f(w) \in L_2$.

Definition:

Let P_1 and P_2 be problems. P_1 is (many-one) reducible to P_2 if there is a TM M computing a function f: $\Sigma_1^* \rightarrow \Sigma_2^*$ such that w is a yes-instance of $P_1 \leftrightarrow$ f(w) is a yes-instance of P_2.

REDUCTION

Definition:

A function $f: \Sigma_1^* \rightarrow \Sigma_2^*$ is a Turing-computable function if there is a Turing machine computing f.

Definition:

Let L_1 and L_2 be languages over Σ_1 and Σ_2, respectively. L_1 is (many-one) reducible to L_2, denoted by $L_1 \leq L_2$, if there is a Turing-computable function $f: \Sigma_1^* \rightarrow \Sigma_2^*$ such that $w \in L_1 \leftrightarrow f(w) \in L_2$.

MEANING OF REDUCTION

P_1 is reducible to P_2 if $\exists$ TM M computing a function f: $\Sigma_1^* \rightarrow \Sigma_2^*$ such that w is a yes-instance of $P_1 \leftrightarrow$ f(w) is a yes-instance of P_2.

- If you can map yes-instances of problem A to yes-instances of problem B, then
  - we can solve A if we can solve B
  - it doesn't mean we can solve B if we can solve A
  - the decidability of B implies the decidability of A

PROPERTIES OF REDUCTION

Theorem: Let L be a language over Σ. $L \leq L$.

Proof:

Let L be a language over Σ.

Let f be an identity function from $\Sigma^* \rightarrow \Sigma^*$.

Then, there is a TM computing f.

Because f is an identity function, $w \in L \leftrightarrow f(w)=w \in L$.

By the definition, $L \leq L$.

PROPERTIES OF REDUCTION

Theorem: Let L_1 and L_2 be languages over Σ.
If $L_1 \leq L_2$, then $\overline{L_1} \leq \overline{L_2}$.

Proof:

Let L_1 and L_2 be languages over Σ.

Because $L_1 \leq L_2$, there is a function f such that $w \in L_1 \leftrightarrow f(w) \in L_2$, and a TM T computing f.

$w \in \overline{L_1} \leftrightarrow f(w) \in \overline{L_2}$.

By the definition, $\overline{L_1} \leq \overline{L_2}$.

PROPERTIES OF REDUCTION

Theorem: Let L_1, L_2 and L_3 be languages over Σ.
If $L_1 \leq L_2$ and $L_2 \leq L_3$, then $L_1 \leq L_3$.

Proof:

Let L_1, L_2 and L_3 be languages over Σ.

There is a function f such that $w \in L_1 \leftrightarrow f(w) \in L_2$, and a TM T1 computing f because $L_1 \leq L_2$.

There is a function g such that $w \in L_2 \leftrightarrow g(w) \in L_3$, and a TM T2 computing g because $L_2 \leq L_3$.

$w \in L_1 \leftrightarrow f(w) \in L_2 \leftrightarrow g(f(w)) \in L_3$, and $T1 \rightarrow T2$ computes $g(f(w))$.

By the definition, $L_1 \leq L_3$.

Using Reduction to Prove Decidability

Theorem: If L_2 is recursive, and $L_1 \leq L_2$, then L_1 is also recursive.

Proof:

Let L_1 and L_2 be languages over Σ, $L_1 \leq L_2$, and L_2 be recursive.

Because L_2 is recursive, there is a TM T_2 computing χ_{L2}.

Because $L_1 \leq L_2$, there is a TM T_1 computing a function f such that $w \in L_1 \leftrightarrow f(w) \in L_2$.

USING REDUCTION TO PROVE DECIDABILITY

Construct a TM $T=T_1 \rightarrow T_2$. We show that T computes χ_{L1}.

- If $w \in L_1$, T_1 in T computes $f(w) \in L_2$ and T_2 in T computes $\chi_{L2}(f(w))$, which is 1.
- If $w \notin L_1$, T_1 in T computes $f(w) \notin L_2$ and T_2 in T computes $\chi_{L2}(f(w))$, which is 0.

Thus, L_1 is also recursive.

USING REDUCTION TO PROVE UNDECIDABILITY

Collorary:

If L_1 is not recursive, and $L_1 \leq L_2$, then L_2 is not recursive.