On Undecidability

Dr. Igor Ivkovic

iivkovic@uwaterloo.ca

[with material from “Introduction to Automata Theory, Languages, and Computation”, by Hopcroft, Motwani, and Ullman, published by Addison Wesley]

Objectives

- Alphabets, Strings, and Languages
- Decidable and Undecidable Computational Problems
- The Diagonalization Language
- Problem Reductions
- (Optional Material) Introduction to Turing Machines

Alphabets, Strings, and Languages /1

- **Σ – An alphabet is any finite set of symbols**
- Examples:
  - ASCII, Unicode, {0,1} (binary alphabet), {a,b,c}
- **A string (a/k/a word) is a finite sequence of symbols chosen from an alphabet Σ**
  - **Length of a string w is usually denoted as |w|**
- The set of strings over an alphabet Σ is the set of lists, each element of which is a member of Σ
  - Strings shown with no commas, e.g., abc

Alphabets, Strings, and Languages /2

- **ε stands for the empty string**
  - **A string of length 0, $|\varepsilon| = 0$**
- **Σ^* denotes the set of all strings for an alphabet**
  - **Σ^+ denotes the set of all strings for an alphabet minus the empty ε string (i.e., $\Sigma^* - \{\varepsilon\}$)**
  - **Σ^k denotes all strings of length k**
- Example:
  - $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, . . . \}$
  - $\Sigma^1 = \{0, 1\}$
  - $\Sigma^2 = \{00, 01, 10, 11\}$
  - …

Alphabets, Strings, and Languages /3

- **A language L is a subset of Σ^* for some alphabet Σ**
  - **That is, if L is a language over Σ, then $L \subseteq \Sigma^*$**
- Example:
  - Strings of 0's and 1's with no two consecutive 1's.
- L = {ε, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, . . . }
  - Two strings of length one
  - Three strings of length two
  - Five strings of length three
  - Eight strings of length four
  - **How many strings of length five?**

Alphabets, Strings, and Languages /4

- Let $L = \{0, 11\}$
  - $L^0 = \{\varepsilon\}$ represents the selection of zero strings from L
  - $L^1 = L$ represents the selection of one string at a time
  - $L^2 = \{00, 011, 110, 1111\}$
  - $L^3 = \{000, 0011, 0110, 01111, 1100, 11011, 11110, 111111\}$
  - …
  - $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
- Also note
  - $\emptyset^0 = \{\varepsilon\}$
  - $\emptyset^i = \{\}$ for any $i > 0$
  - $\emptyset^* = \{\varepsilon\}$

Alphabets, Strings, and Languages /5

- Additional properties of languages:
  - Alphabets Σ are finite but the languages derived from such alphabets may contain infinitely many strings
  - ∅ represents the empty language, which is also a language over any alphabet Σ
- **Language Membership Problem:**
  - **Given a string w from Σ*, decide if w is in L**
  - **This is also a decision problem that returns "true" or "false" (alternatively "yes" or "no") as its answer**
- Example:
  - L = {w | w is string made up of an equal number of 0s and 1s}
  - Is 01100 part of L?
    - Obviously not but how would you prove it systematically?

Alphabets, Strings, and Languages /6

- **Why bother with languages and language membership?**
  - Strings can be assigned additional semantics that go beyond just 0s and 1s
  - For instance, programming expressions, scheduling problems represented as graphs, logical expressions
  - **With additional semantics, solving a problem could be viewed as verifying that a problem instance (a string) belongs to a particular language that defines all problem instances that have the desired property**
  - That is, by utilizing language formalisms we can simplify the problem-solving of complex mathematical problems that are otherwise difficult to model and understand

Formal Proofs as Enumerable Strings

- **A formal proof is a sequence of logical expressions, each of which follows from the ones before it**
  - One can encode logical expressions in Unicode text
  - Convert Unicode text expression to binary strings, with 8 or 16 bits/character
  - And then represent binary strings as the corresponding integer values
  - **<u>An enumeration of a set is a one-to-one correspondence between the set and the positive integers</u>**
  - **Hence, we can enumerate the set of formal proofs**
  - Similarly, we can enumerate programs

Are All Languages Enumerable?

- **For instance, are the languages over {0,1}* countable?**
- Proof:
  - Suppose one could enumerate all languages over {0,1}* and assign i to be the index value for "the i-th language"
  - Consider the language L = { w | w is the i-th binary string and w is not in the i-th language} and L $\subseteq$ {0,1}*
  - Since L is in {0,1}* then let L be the j-th language for some particular j and let x be the j-th string in {0,1}*
  - **If x is in L then x is not in L by the definition of L**
  - **If x is not in L then x is in L by the definition of L**
  - **Both are contradictions!**
  - The starting assumption that there was an enumeration of the languages in {0,1}* is wrong
  - **Hence, there are more languages than programs and there are languages with no membership algorithm**

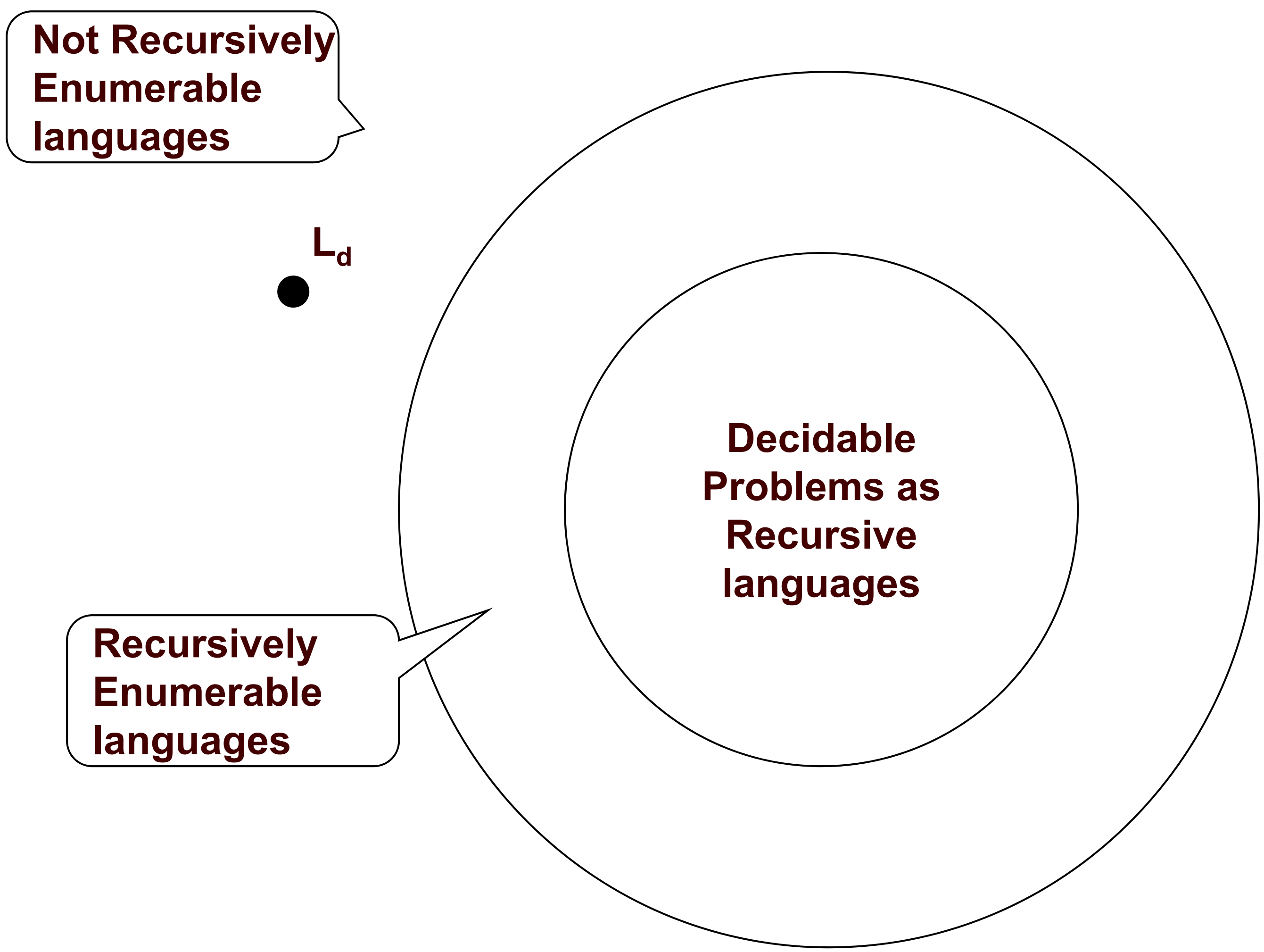
Decidable Computational Problems

- **A computational problem is decidable if there is a formal procedure (e.g., an algorithm) to answer it**
- **Example:**
  - Given a string w from Σ^*, decide if w is in L
  - **If there is a computable function f such that f(w) = 1 if w $\in$ L and f(w) = 0 if w $\notin$ L then L is recursive (and decidable)**
  - **If there is only a computable function g(w) such that g(w) = 1 if w $\in$ L and g is unknown/undefined otherwise then L is recursively enumerable (and undecidable)**

Recursively Enumerable Languages

- **A decidable problem can be formally referred to as the recursive language**
  - Regular languages and context-free languages are recursive and hence decidable
  - **The Recursively Enumerable (RE) and not Recursively Enumerable languages represent undecidable problems**
- **Decidable Sets and First-Order Logic:**
  - A set S is decidable/recursive if there is a formula $\varphi(x)$ such that $\vdash \varphi(t)$ for $t \in S$ and $\vdash \neg\varphi(t)$ for $t \notin S$
- **Theorem. Validity is Undecidable:**
  - The set VALID = $\{ \ulcorner\varphi\urcorner \mid \vdash \varphi\}$ is not recursive, where $\ulcorner\varphi\urcorner$ represents an enumeration/coding of formulas φ

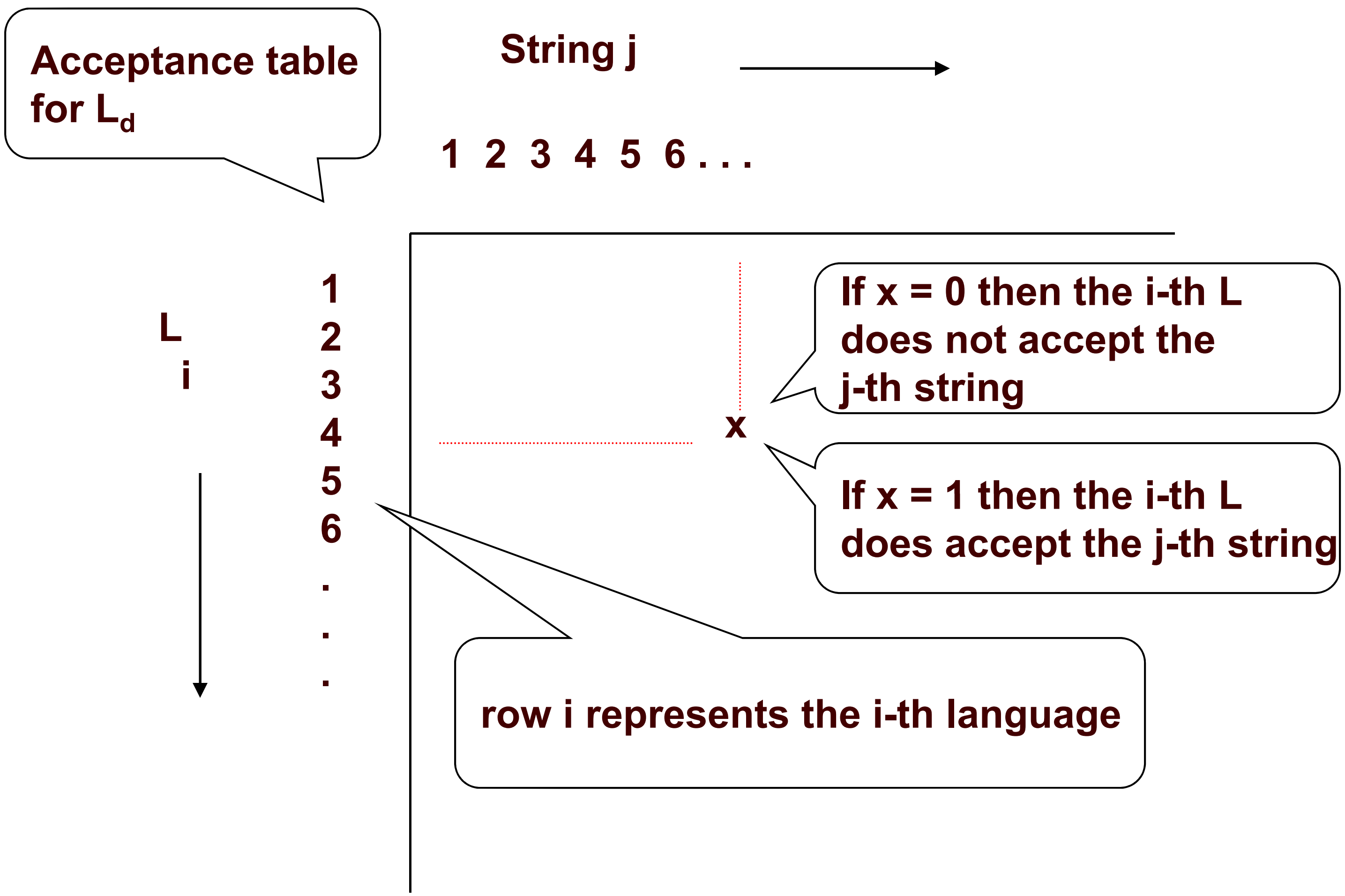
Decidable Languages Hierarchy



The Diagonalization Language /1

- Recall our attempt to enumerate all possible languages
  - Let us define a language based on that discussion, where w is the i-th string and the i-th language does not hold w
  - **Let us call this language the diagonalization language L_d**
- Interesting fact about L_d:
  - **L_d is not recursively enumerable since we cannot find a function g for which g(w) = 1 if $w \in L_d$**
  - Let us illustrate and reason this fact using the following acceptance table

The Diagonalization Language /2



The Diagonalization Language /3

- **An acceptance table like the one on the previous slide can be "diagonalized":**
  - Construct a string D by complementing each bit along the major diagonal
  - Let $D = a_1a_2 \ldots$ where $a_i = 0$ if the entry at (i, i) is 1, and $a_i = 1$ if the entry at (i, i) is 0
- **Could D be a row of the table, representing the language accepted by some L?**
  - Let us assume that D is the j-th row
  - However, D disagrees with the j-th row at the j-th column
  - Hence, D is not a row and is not accepted by any L

Problem Reduction

- **Definition. Problem Reduction:**
  - For two decision problems A over Σ and B over Δ, we say that A can be recursively reduced to B, denoted as $A \leq B$, if there is a computable function $f: \Sigma^* \rightarrow \Delta^*$ such that for $\forall x \in \Sigma^*$ it holds that $x \in A \leftrightarrow f(x) \in B$
- **Let us assume that A can be reduced to B:**
  - If A is undecidable then so is B
  - If B is recursive/decidable then so is A
- **Halting(P):**
  - Given a program P and an input x, decide if P terminates on input x
  - **This problem is undecidable**
    - Recall our explanation using Schrödinger's Cat

Post's Correspondence Problem (PCP)

- **Post's Correspondence Problem (PCP) Introduced:**
  - Example of a problem that does not mention TMs in its statement, yet is undecidable
  - From PCP, we can prove that many other non-TM problems are undecidable
- **An instance of PCP is a list of pairs of nonempty strings over some alphabet Σ**
  - Such as $(w_1, x_1), (w_2, x_2), \ldots, (w_n, x_n)$
  - The answer to this instance of PCP is "true" iff there exists a nonempty sequence of indices i1, … ik, such that $w_{i1} \ldots w_{ik} = x_{i1} \ldots x_{ik}$

Example: PCP

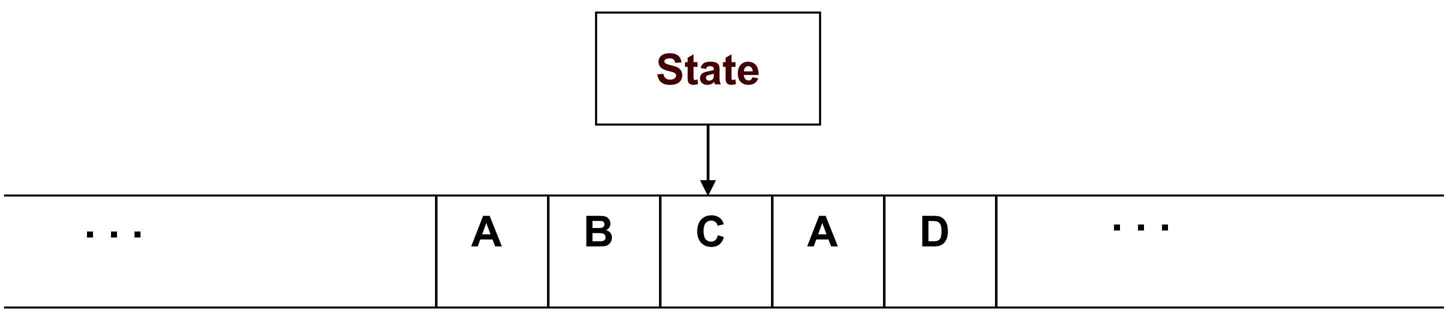
- Let the alphabet be {0, 1}
  - **Let the PCP instance consist of the two pairs (0, 01) and (100, 001)**
  - There is no solution to this problem
  - Cannot start with 1000 and 00101 since the first characters are different
  - However, if one starts with 0 100 and 01 001, the two strings still cannot be made equal
  - For example, 0 100 100 and 01 001 001 comes close but adding more 1s to the first string always requires two zeros while the second string always gets a 1 at the end

Modified PCP (MPCP)

- **Suppose we add a third pair, so the instance is: (1) = (0, 01); (2) = (100, 001); (3) = (110, 10).**
  - Now (1)(3) is a solution; both strings are 0110.
  - That is, any sequence in (1)(2)*(3) is a solution
- **Modified PCP (MPCP) problem:**
  - Equivalent to PCP but the solution must start with the first pair in the list
  - Useful for proving that PCP is undecidable
- **Interesting Consideration:**
  - Reduce PCP to a decision problem of predicate logic

(Optional Material) Turing Machines

- The purpose of the theory of Turing Machines (TM) is to prove that certain specific languages have no algorithm
  - Start with a language about Turing Machines themselves
  - Reductions are used to prove more common questions undecidable
  - **Action: Based on the state and the tape symbol under the head (1) change state, (2) rewrite the symbol, and (3) move the head one square**



Infinite tape with squares containing tape symbols chosen from a finite alphabet

Why Turing Machines?

- Why not deal with C programs or similar?
  - **Answer: One can but using TMs is simpler yet TMs are as powerful as any computer**
  - **And with the infinite amount of memory**
- Why not use Finite Automata?
  - Programming models are not built with a limit on memory
  - In practice, we can always add more memory
- **Church-Turing Thesis:**
  - Any real-world computation can be translated into an equivalent computation involving TMs
  - That is, Turing Machines are as computationally capable as any real-world computation problem

Turing Machine (TM) Formalism

- **A TM is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where**
  - Q is a finite set of states
  - Σ is an input alphabet
  - Γ is a tape alphabet (typically contains Σ)
  - δ is a transition function
  - $q_0 \in Q$ is the start state
  - $B \in \Gamma - \Sigma$ is the blank symbol
    - The entire TM except for the input is blank initially
  - $F \subseteq Q$ is the set of final states
- TM notation conventions are similar to FA conventions
  - a, b, … are input symbols
  - …, X, Y, Z are tape symbols
  - …, w, x, y, z are strings of input symbols
  - α, β,… are strings of tape symbols

TM Transition Function δ

- **The TM transition function δ takes two arguments:**
  - **A state in Q**
  - **A tape symbol in Γ**
  - $\delta(q, Z)$ is either undefined or a triple of the form (p, Y, D)
  - p is a state
  - Y is the new tape symbol
  - D is a direction, L or R
- If $\delta(q, Z) = (p, Y, D)$ then in state q scanning Z under its tape head, the TM will do the following:
  - **Changes the state to p**
  - **Replaces Z by Y on the tape**
  - **Moves the head one square in direction D**
    - **D = L implies move left; D = R implies move right**

Example: Turing Machine

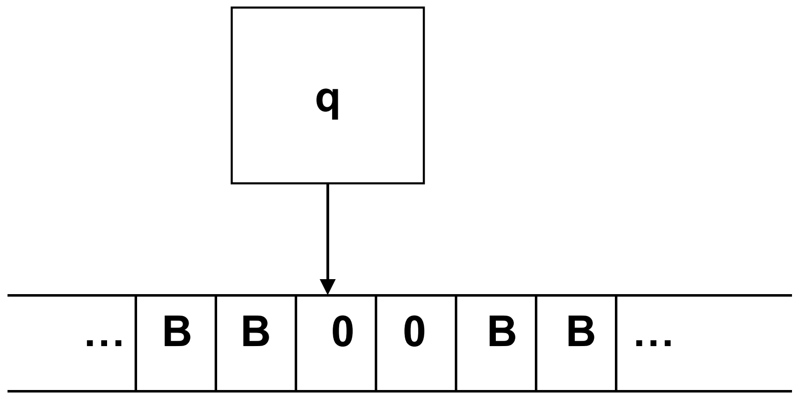
- Example:
  - The TM scans its input right, looking for a 1
  - If it finds one, it changes it to a 0, goes to the final state f, and then halts
  - If it reaches a blank, it changes it to a 1 and moves left
  - States = {q (start), f (final)}
  - Input symbols = {0, 1}
  - Tape symbols = {0, 1, B}
  - $\delta(q, 0) = (q, 0, R)$
  - $\delta(q, 1) = (f, 0, R)$
  - $\delta(q, B) = (q, 1, L)$

TM Simulation /1

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

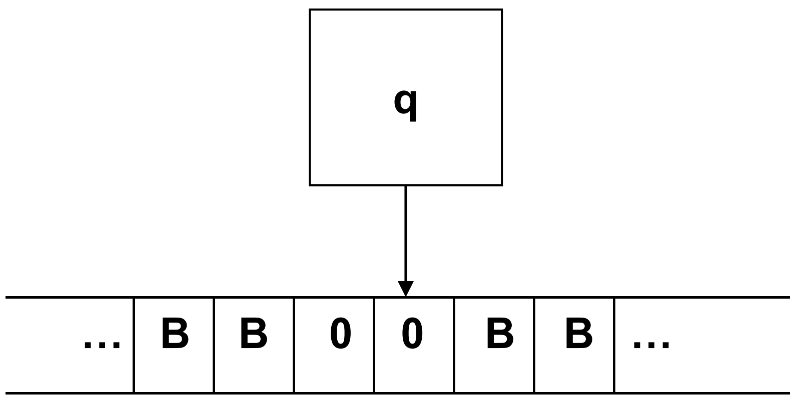


TM Simulation /2

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

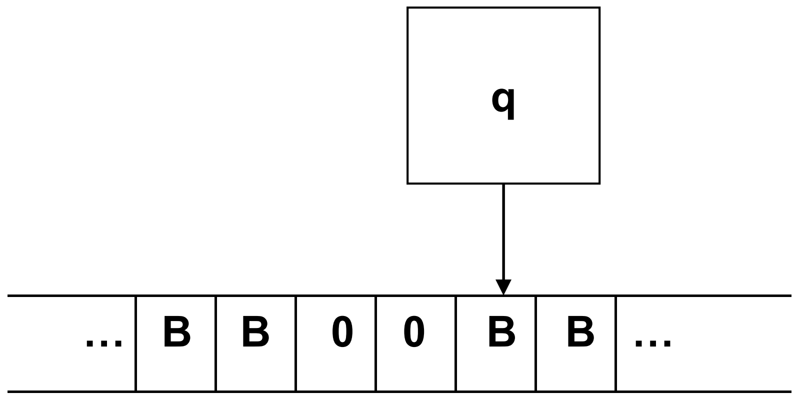


TM Simulation /3

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

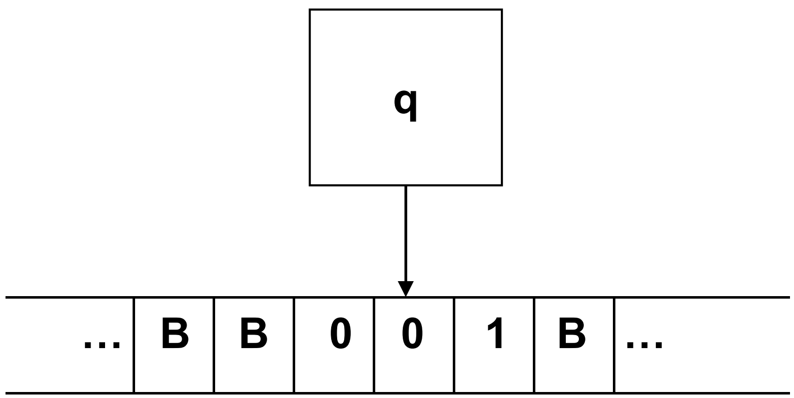


TM Simulation /4

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

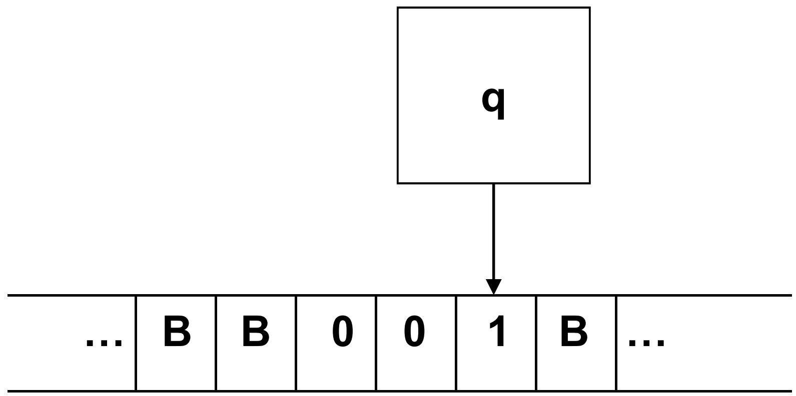


TM Simulation /5

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

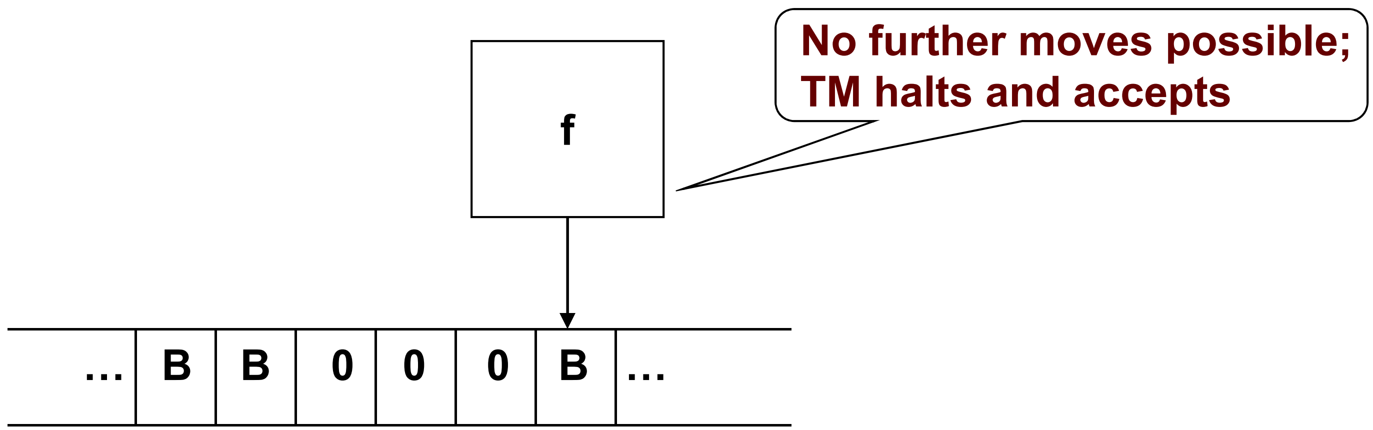


TM Simulation /6

δ(q, 0) = (q, 0, R)

δ(q, 1) = (f, 0, R)

δ(q, B) = (q, 1, L)



TM Moves and Languages

- Formal definition of the δ transition function
  - If $\delta(q, Z) = (p, Y, R)$ then $\alpha q Z \beta \vdash \alpha Y p \beta$
  - If Z is the blank symbol B then also $\alpha q \vdash \alpha Y p$
  - If $\delta(q, Z) = (p, Y, L)$ then for any X $\alpha X q Z \beta \vdash \alpha p X Y \beta$
  - In addition, it also holds that $q Z \beta \vdash p B Y \beta$
- **A TM defines a language by its final states**
  - $L(M) = \{w \mid q_0 w \vdash^* I$, where I is an ID with a final state$\}$
- **Alternatively, a TM can accept a language by halting**
  - $H(M) = \{w \mid q_0 w \vdash^* I$, where I is an ID from which there are no more moves possible$\}$
- If L = L(M) then there is a TM M' such that L = H(M')
- If L = H(M) then there is a TM M" such that L = L(M")

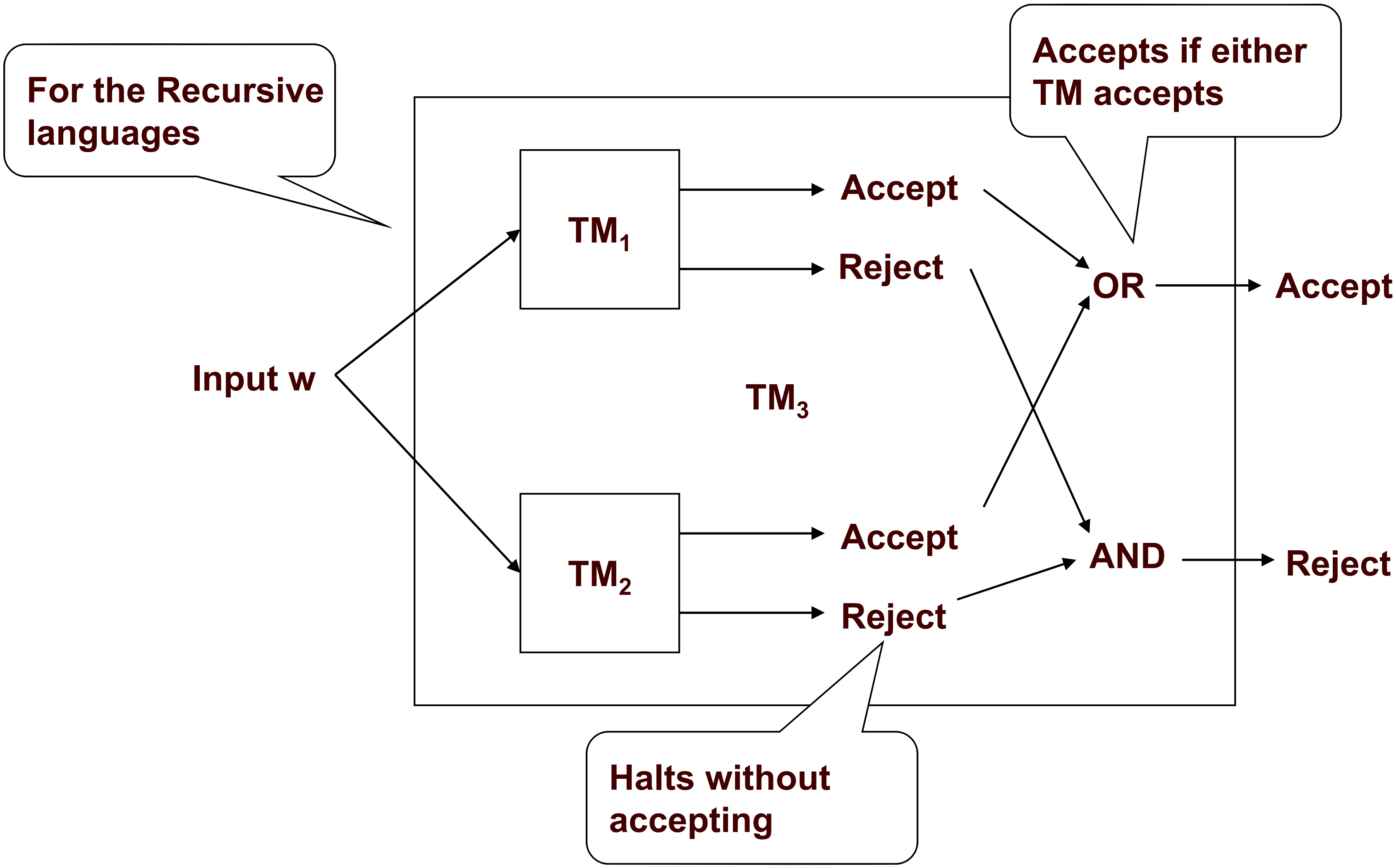
TM Accepting and TM Halting Equivalences

- **TM Accepting to TM Halting:**
  - Modify M to become M’ as follows:
  - For each accepting state of M, remove any moves so M’ halts in that state
  - Ensure that M’ cannot accidentally halt:
  - Introduce a new state s, which runs to the right forever; that is $\delta(s, X) = (s, X, R)$ for all symbols X
  - If q is not accepting and $\delta(q, X)$ is undefined, let $\delta(q, X) = (s, X, R)$
- **TM Halting to TM Accepting:**
  - Modify M to become M” as follows:
  - Introduce a new state f, the only accepting state of M”
  - Ensure that f has no moves
  - If $\delta(q, X)$ is undefined for any state q and symbol X, define it by $\delta(q, X) = (f, X, R)$.

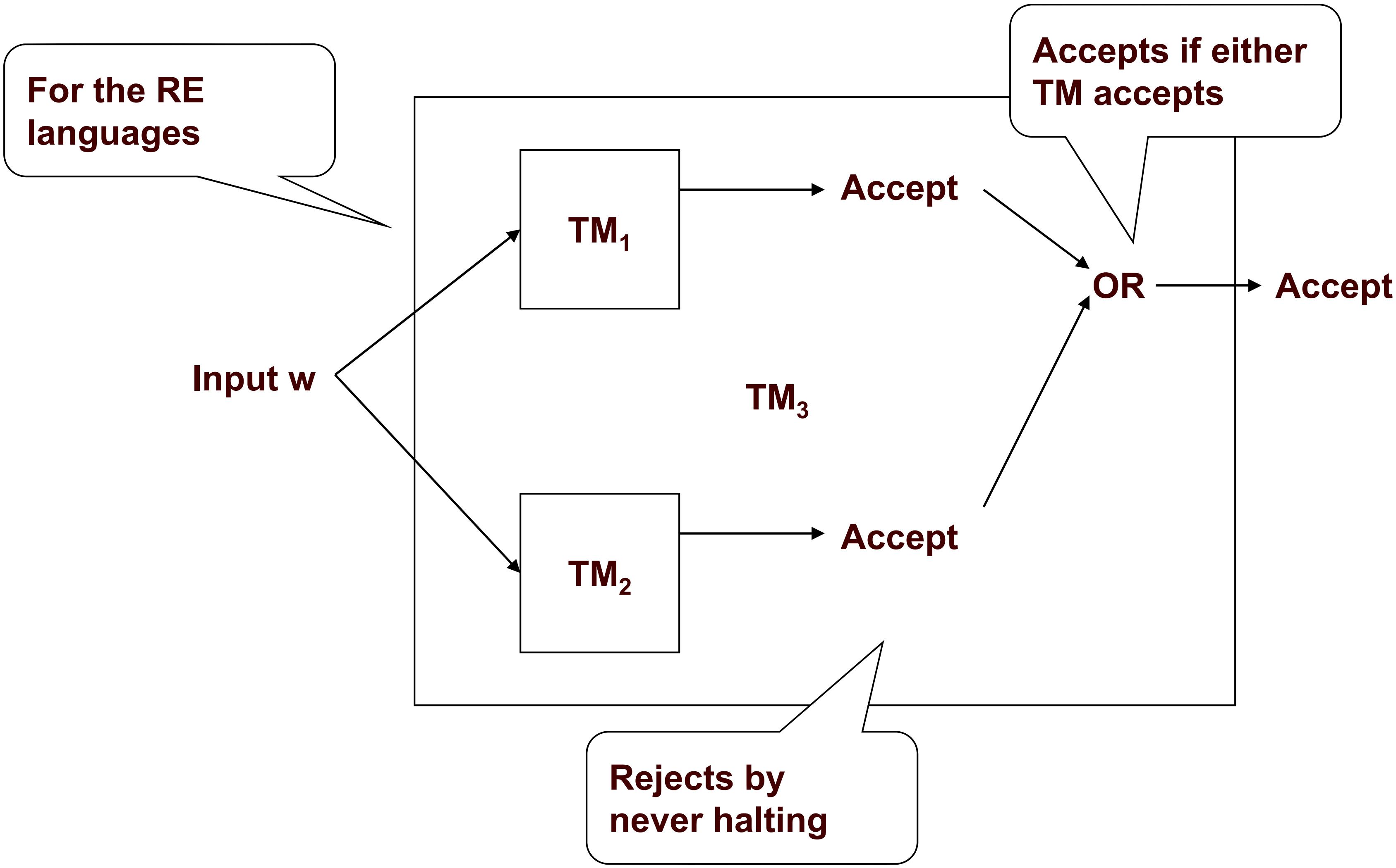
Recursively Enumerable Languages

- **As demonstrated, the classes of languages defined by TMs using final states and halting are the same**
  - **This class of languages are the recursively enumerable languages**
- **Furthermore, an algorithm is a TM that is guaranteed to halt whether or not it accepts**
  - If L = L(M) for some TM M that is an algorithm then L is a recursive/decidable language

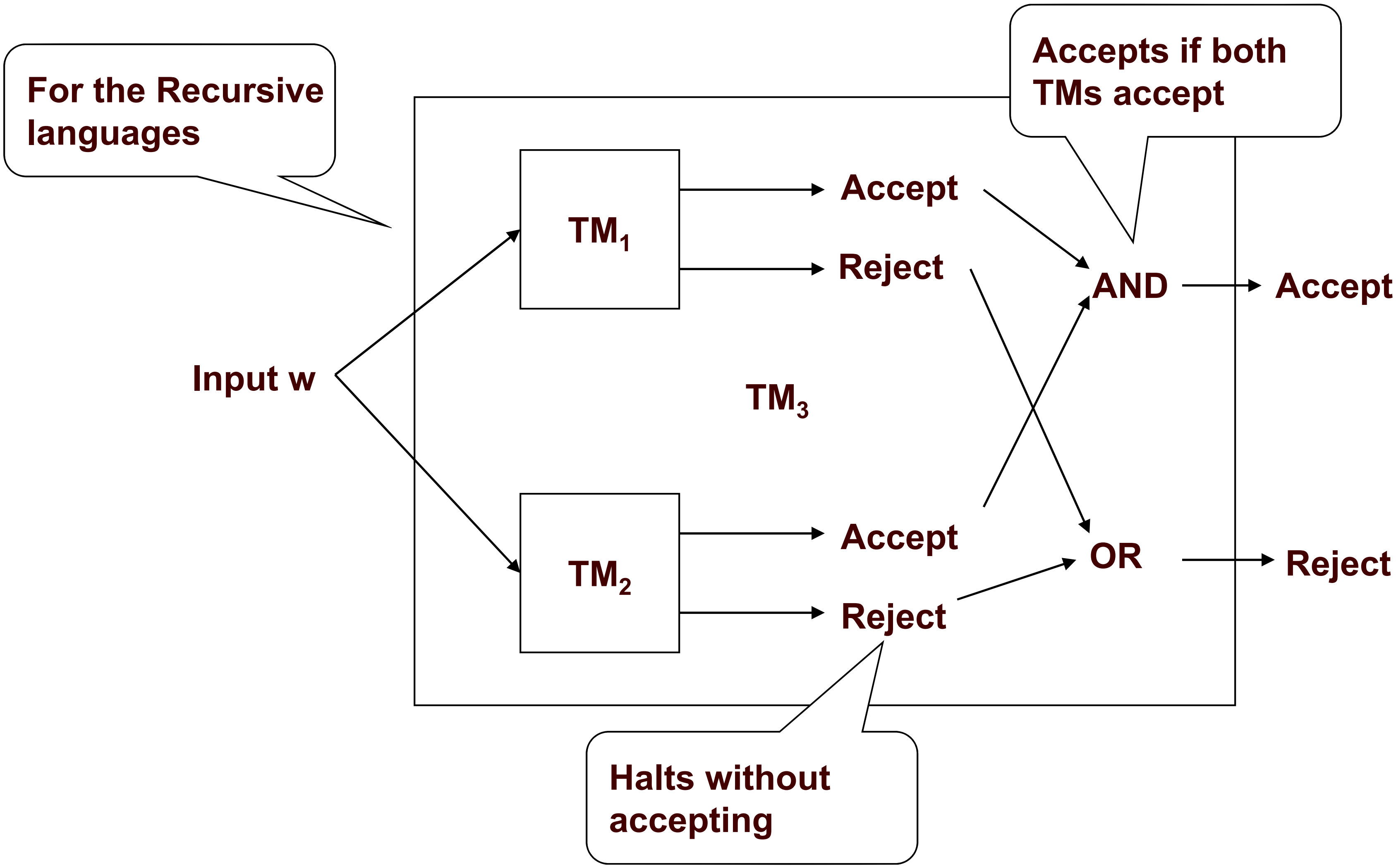
TM Closure Under Union Illustrated /1



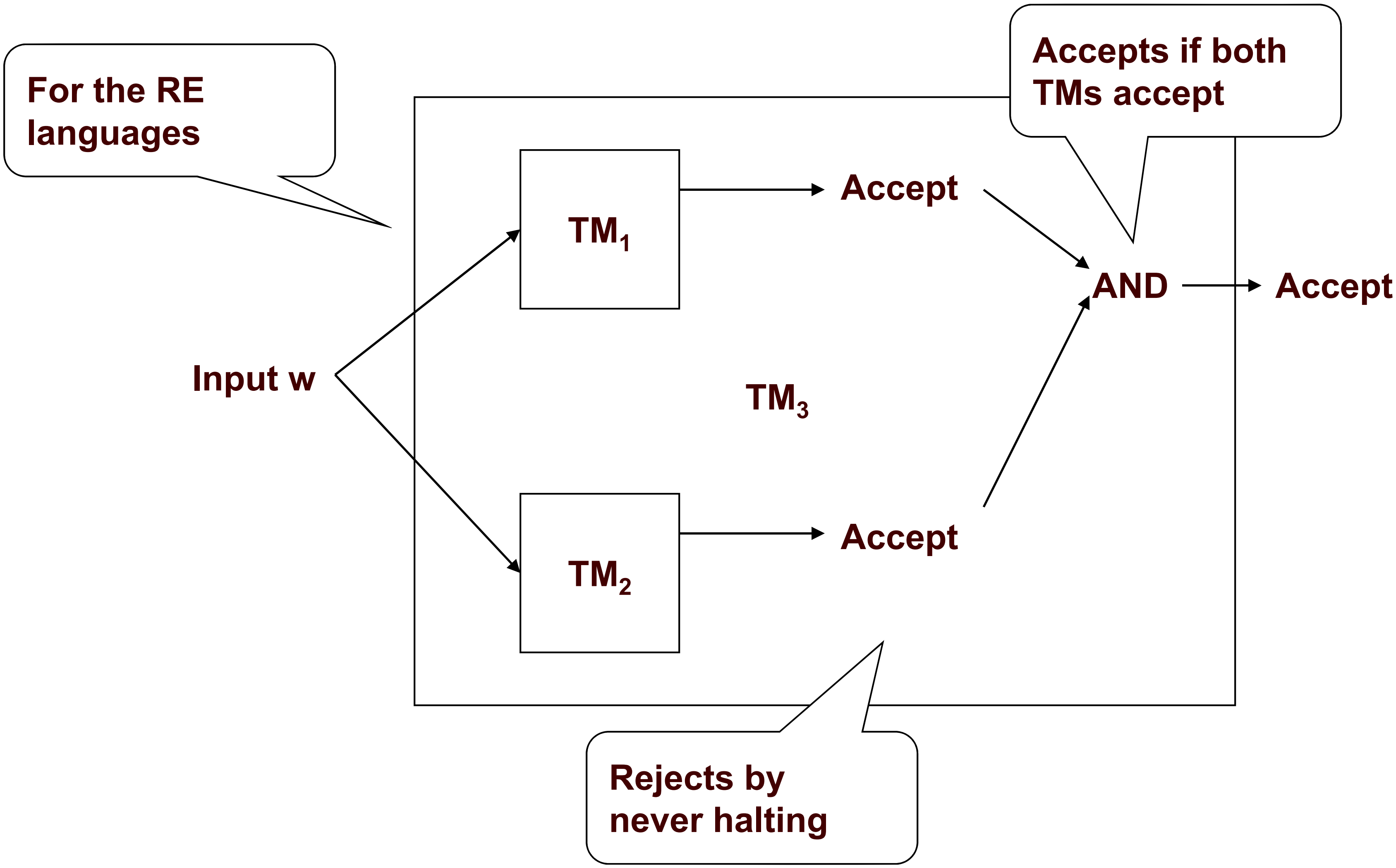
TM Closure Under Union Illustrated /2



TM Closure Under Intersection Illustrated /1



TM Closure Under Intersection Illustrated /2



Food for Thought

- **Answer Assignment #5 questions**
  - Assignment #5 includes several practice exercises related to Undecidability
- **The Final Exam will cover:**
  - Lecture Notes #1 to #11
    - The emphasis will be on Lecture Notes #6 to #11
  - "Food for Thought" Readings and Exercises
  - Assignments #1 to #5
    - The emphasis will be on Assignment #3 to #5

Closing Remarks

- The problems of language here are really serious. We wish to speak in some way about the structure of the atoms. But we cannot speak about atoms in ordinary language. – Werner Heisenberg

- Logic will get you from A to B. Imagination will take you everywhere. – Albert Einstein

- **Contrariwise, if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic. – Lewis Carroll**

- **One of the secrets of life is that all that is really worth the doing is what we do for others. – Lewis Carroll**

[BrainyQuote. Online, 2012. http://www.brainyquote.com]