Chapter 23: Turing Machine Languages

I. Theory of Automata

II. Theory of Formal Languages

→ III. Theory of Turing Machines …

Equivalent Variations of the Turing Machines:

1. Non deterministic TM = deterministic TM
2. nPDA = Push Down Automatas with n stacks.
   2PDA = nPDA = TM for all $n \geq 2$
3. Turing Machines with n tapes ($n \geq 2$) et n tape heads has the same capacity as a Turing Machine with one tape head.
4. Turing Machines with a tape infinite to the left and to the right has the same capacity as a Turing Machines with a tape finite to the left but infinite to the right.

- Definition. A language L over the alphabet S is recursively enumerable if there exists a Turing machine such that for every word $w \in L$, w is accepted, and for every word $w \notin L$, either w is rejected (crashes) or w causes the machine to go into an infinite loop.
- Definition. A language L over an alphabet S is recursive there exists a Turing machine such that for every word $w \in L$, w is accepted, and for every word $w \notin L$, w is rejected (crashes).

- Theorem. If there exist two Turing machines T_1 and T_2 that accept languages L_1 and L_2, then there exists a Turing machine that accepts $L_1 + L_2$.
- Proof: (abbreviated) It is possible to build a Turing machine that simulates running the input on the two machines alternately.
  - First, transform T_1 which accepts L_1 to a Turing machine such that for every word that is not in L_1, the machine loops forever. Call the new machine T_1'. Do the same for L_2.

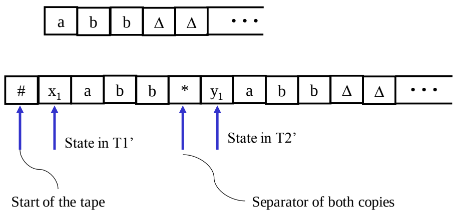
Insert a special character # in the first cell. Whenever the character # is read the machine crashes. Our machine now crashes in only one situation: when there are no transitions for a read character.

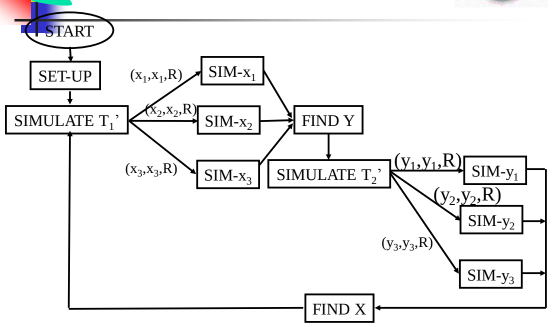
For every state and every character x that has no existing exit edge we add a new transition (x, x, R) going to a new state SMWHERE.

From the state SMWHERE we will always get into an infinite loop by putting transitions that stay in SMWHERE and of the form (y, y, R) for every character y.

- The initial steps of the new Turing machine must make 2 copies of the input word on the tape. (See next page.)
- A SIMULATE-T_1' state in T_3 begins simulation of the next step of T_1'.
- For each state x_i of T_1', a set of states of T_3 performs the simulation of the a step of execution starting from x_i.
- A state FIND-Y pushes the tape head right until it finds a state in T_2'.
- Similarly, SIMULATE-T_2', states for simulating the execution from each state in T_2', and a state FIND-X must be added to T_3.

SIMULATE-T_1' and SIMULATE-T_2'





The machine T_3 will simulate both machines T_1' and T_2' . None of the two machines will crash.

If the word in in L_1, then T_2' will either accept the word or get into an infinite loop and T_1' will have time to accept the word. So T_3 will accept the word.

If the word is in L_2, then T_1' will either accept the word or get into an infinite loop and T_2' will have time to accept the word. So T_3 will accept the word.

The Turing machine T_3 accepts the language $L_1 + L_2$.

- If a language L and its complement L' are both recursively enumerable then L is recursive.

- Proof: (abbreviated)

T_1 a Turing machine for L.

T_2 a Turing machine for L'.

We transform both machines into new Turing machines T_1'and T_2'such that:

T_2'rejects every word in L' and gets into an infinite loop for every word in L.

T_1'accepts every word in L and gets into an infinite loop for every word in L'.

The machine T_3 will simulate both machines T_1' and T_2'(as it was done for the union). None of the two machines will crash.

If the word in in L, then T_2'will get into an infinite loop and T_1' will have time to accept the word. So T_3 will accept the word.

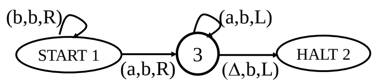
If the word is not in L, then T_1' will get into an infinite loop and it will give time for T_2'to reject the word. So T_3 will reject the word.

The Turing machine T_3 makes the language L recursive.

- Theorem. If there exist two Turing machines T_1 and T_2 that accept languages L_1 and L_2, then there exists a Turing machine that accepts $L_1 \cap L_2$.

- Remark: The complement of a recursively enumerable language is not necessarily recursively enumerable.

- The Encoding of Turing machines: Example



From	To	Read	Write	Move
1	1	b	b	R
1	3	a	b	R
3	3	a	b	L
3	2	Δ	b	L

From	To	Read	Write	Move
X_1	X_2	X_3	X_4	X_5

Encoding a state X1,X2 (positive integers): $a^{X1}ba^{X2}b$:

X_3,X_4	Code
a	aa
b	ab
Δ	ba
#	bb

X_5	Code
L	a
R	b

Code of the row: abaaabaaabb

From	To	Read	Write	Move
1	3	a	b	R

From	To	Read	Write	Move	Code
1	1	b	b	R	ababababb
1	3	a	b	R	abaaabaaabb
3	3	a	b	L	aaabaaabaaaba
3	2	Δ	b	L	aaabaabbaaba

Code of the machine:
ababababbabaaabaaabbaaabaaabaaabaaaabaabbaaba

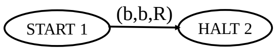
Code Word Language: CWL = language($\mathbf{(a^{+}ba^{+}b(a+b)^{5})^{*}}$)

Remark: It is possible to determine if a word in CWL is the code of a Turing machine.

ALAN = all words w∈CWL that are not accepted by the Turing machines that they represent.

ALAN⊂CWL

Example:



From	To	Read	Write	Move
1	2	b	b	R

Language L for this machine: all words that start with b

Code of the Turing machine: abaabababb

abaabababb ∉L therefore abaabababb ∈ALAN

- Example: The code word for a machine that accepts language(**(a+b)***) is not in ALAN.
- Example: The code word for a machine that accepts the empty language is in ALAN.
- Example: The code word for a machine that accepts L=language(**(a+b)*aa(a+b)***) contains aa, and thus is in L. Thus the code word is not in ALAN.

- Theorem. There does not exist any Turing machine that accepts ALAN.
- Proof: Assume there is a Turing machine T that accepts ALAN. We denote the code word for T as code(T). Either code(T)∈ALAN, or code(T)∉ALAN.
  - Case 1. code(T)∈ALAN. By definition of T, code(T)∉ALAN. A contradiction.
  - Case 2. code(T)∉ALAN. By definition of T, code(T) is not accepted by T. By definition of T, code(T)∈ALAN. A contradiction.

Thus there is no Turing machine that accepts ALAN.

- Theorem. Not all languages are recursively enumerable.

- Definition. A universal Turing machine is a Turing machine MTU such that:
  - Input words to MTU have the form:

    #w#x

    where w is the code word that represents a Turing machine T and x is a word containing letters of T's input alphabet.
  - MTU will operate on the data #w#x exactly the same as T would operate on x. (MTU crashes, accepts, or loops if and only if T does the same.)
- Remark: Universal Turing machines exist.

The Halting Problem:

Does there exist a Turing machine such that given an input word w and a code word for a Turing machine T that can determine whether on not T halts (enters a HALT state) on input w?

- Theorem. No Turing machine exists that can solve the halting problem.

- Proof idea: If we assume such a machine exists, we can build a Turing machine that accepts ALAN.