Turing Machines and Effective Computability

Turing machines

- the most powerful automata (> FAs and PDAs)
- invented by Turing in 1936
- can compute any function normally considered computable
- Turing-Church Theses:
  - Anything (function, problem, set etc.) that is (though to be) computable is computable by a Turing machine (i.e., Turing-computable).
- Other equivalent formalisms:
  - post systems (string rewriting system)
  - PSG (phrase structure grammars) : on strings
  - μ-recursive function : on numbers
  - λ-calculus, combinatory logic: on λ-term
  - C, BASIC, PASCAL, JAVA languages,… : on strings

Informal description of a Turing machine

1. Finite automata (DFAs, NFAs, etc.):
   - limited input tape: one-way, read-only
   - no working-memory
   - finite-control store (program)
2. PDAs:
   - limited input tape: one-way, read-only
   - one additional stack as working memory
   - finite-control store (program)
3. Turing machines (TMs):
   - a semi-infinite tape storing input and supplying additional working storage.
   - finite control store (program)
   - can read/write and two-way(move left and right) depending on the program state and input symbol scanned.

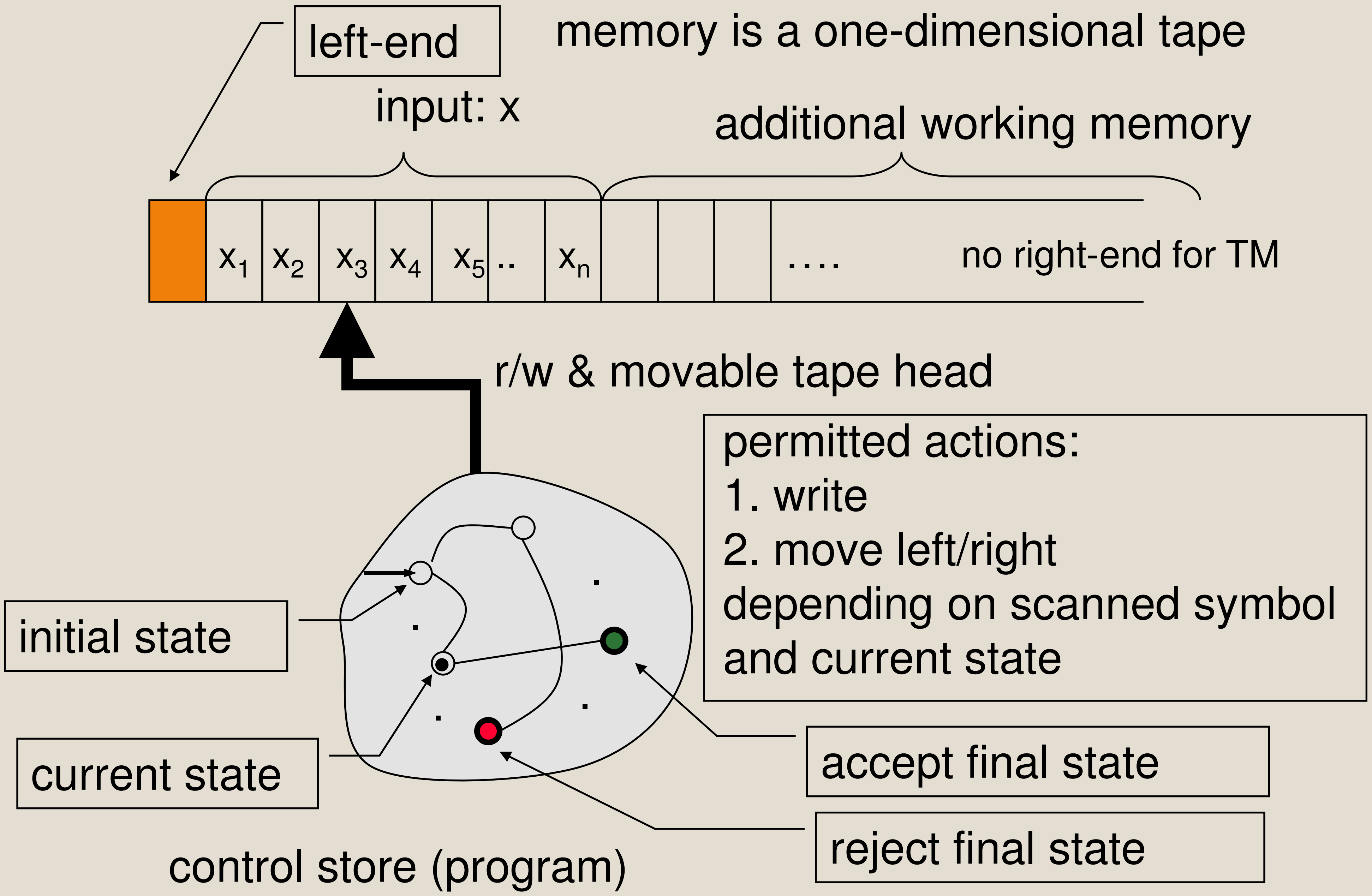
Turing machines and LBAs

4. Linear-bounded automata (LBA): special TMs
   - the input tape is of the same size as the input length (i.e., no additional memory supplied except those used to store the input)
   - can read/write and move left/right depending on the program state and input symbol scanned.

- Primitive instructions of a TM (like +,-,*, etc in C or BASIC):
  1. L, R // moving the tape head left or right
  2. $a \in \Gamma$, // write the symbol $a \in \Gamma$ on the current scanned position

  depending on the precondition:
  1. current state and
  2. current scanned symbol of the tape head

The model of a Turing machine



The structure of a TM instruction:

- An instruction of a TM is a tuple:

  $$(q, \quad a, \quad p, \quad d) \in Q \times \Gamma \times Q \times (\Gamma \cup \{L,R\})$$

  where
  - q is the current state
  - a is the symbol scanned by the tape head
  - (q,a) define a precondition that the machine may encounter
  - (p,d) specify the actions to be done by the TM once the machine is in a condition matching the precondition (i.e., the symbol scanned by the tape head is 'a' and the machine is at state q)
  - p is the next state that the TM will enter
  - d is the action to be performed:
    - $d = b \in \Gamma$ means "write the symbol b to the tape cell currently scanned by the tape head".
    - d = R (or L) means "move the tape head one tape cell in the right (or left, respectively) direction.
- A Deterministic TM program δ is simply a set of TM instructions (or more formally a function: $\delta: Q \times \Gamma \dashrightarrow Q \times (\Gamma \cup \{L,R\})$)

Formal Definition of a standard TM (STM)

- A deterministic 1-tape Turing machine (STM) is a 9-tuple

  $M = (Q,\Sigma,\Gamma, [, \square, \delta, s, t,r\ )$ where

  - Q : is a finite set of (program) states with a role like labels in traditional programs
  - Γ : tape alphabet
  - $\Sigma \subset \Gamma$: input alphabet
  - $[\in \Gamma - \Sigma$: The left end-of-tape mark
  - $\square \in \Gamma - \Sigma$ is the blank tape symbol
  - $s \in Q$: initial state
  - $t \in Q$: the accept state
  - $r \neq t \in Q$: the reject state and
  - δ: (Q - {t,r})x Γ --> Qx(Γ U {L,R}) is a *total* transition function with the restriction: if $\delta(p, [\ ) = (q, d)$ then d = R. i.e., the STM cannot write any symbol at left-end and never move off the tape to the left.

Configurations and acceptances

- Issue: h/w to define configurations like those defined at FAs and PDAs ?
- At any time t_0 the TM M's tape contains a semi-infinite string of the form

  $$Tape(t_0) = [\ y_1 y_2 \ldots y_m \square\square\square\square \ldots\ldots \quad (y_m \neq \square)$$
- Let $\square^\omega$ denotes the semi-infinite string:

  $$\square\square\square\square\square \ldots\ldots$$

Note: Although the tape is an infinite string, it has a finite canonical representation: y, where $y = [\ y_1 \ldots y_m$ (with $y_m \neq \square$)

A configuration of the TM M is a global state giving a snapshot of all relevant info about M's computation

Formal definition of a configuration

Def: a cfg of a STM M is an element of

$CF_M =_{def} Q \times \{ [y \mid y \in (\Gamma - \{[\})^*\} \times N$ // N = {0,1,2,…} //

When the machine M is at cfg (p, z, n) , it means M is

1. at state p
2. Tape head is pointing to position n and
3. the input tape content is z.

Obviously cfg gives us sufficient information to continue the execution of the machine.

Def: 1. [Initial configuration:] Given an input x and a STM M, the initial configuration of M on input x is the triple:

(s, [x, 0)

2. If cfg1 = (p, y, n), then cfg1 is an accept configuration if p = t (the accept configuration), and cfg1 is an reject cfg if p = r (the reject cfg). cfg1 is a halting cfg if it is an accept or reject cfg.

One-step and multi-step TM computations

- one-step Turing computation ($|\text{--}_M$) is defined as follows:
- $|\text{--}_M \subseteq CF_M^2$ s.t.

  0. (p,z,n) $|\text{--}_M$ $(q,s^n{}_b(z), n)$ if $\delta(p,z_n) = (q, b)$ where $b \in \Gamma$
  1. (p,z,n) $|\text{--}_M$ $(q,z, n-1)$ if $\delta(p,z_n) = (q, L)$
  2. (p,z,n) $|\text{--}_M$ $(q,z, n+1)$ if $\delta(p,z_n) = (q, R)$
     - where $s^n{}_b(z)$ is the resulting string with the n-th symbol of z replaced by 'b'.
     - ex: $s^4{}_b($ [baa$\underline{a}$cabc) = [baa$\underline{b}$cabc
     - $s^6{}_b($ [baa) = [baa□□b
- $|\text{--}_M$ is defined to be the set of all pairs of configurations each satisfying one of the above three rules.

Notes: 1. if $C=(p,z,n)$ $|\text{--}_M$ (q,y,m) then $n \geq 0$ and $m \geq 0$ (why?)

2. $|\text{--}_M$ is a function [from nonhalting cfgs to cfgs] (i.e., if C $|\text{--}_M$ D & C $|\text{--}_M$ E then D=E).

3. define $|\text{--}^n{}_M$ and $|\text{--}^*{}_M$ (ref. and tran. closure of $|\text{--}_M$) as

Accepting and rejecting of TM on inputs

- x $\in \Sigma$ is said to be accepted by a STM M if
  icfg$_M$(x) $=_{def}$ (s, [x, 0) $|\text{--}*_M$ (t,y,n) for some y and n
  - I.e, there is a finite computation
    (s, [x, 0) = C_0 $|\text{--}_M$ C_1 $|\text{--}_M$ …. $|\text{--}_M$ C_k = (t,y,n)

  starting from the initial configuration and ending at an accept configuration.
- x is said to be rejected by a STM M if
  (s, [x, 0) $|\text{--}*_M$ (r,y,n) for some y and n
  - I.e, there is a finite computation
  - (s, [x, 0) = C_0 $|\text{--}_M$ C_1 $|\text{--}_M$ …. $|\text{--}_M$ C_k = (t,y,n)
  - starting from the initial configuration and ending at a reject configuration.

Notes: 1. It is impossible that x is both accepted and rejected by a STM. (why ?)

Languages accepted by a STM

Def:

1. M is said to *halt* on input x if either M accepts x or rejects x.
2. M is said to *loop* on x if it does not halt on x.
3. A TM is said to be *total* if it halts on all inputs.
4. The language accepted by a TM M,

   $L(M) =_{def} \{x \text{ in } \Sigma^* \mid x$ is accepted by M, i.e., $(s, [x□^{\omega}, 0) \mid\text{--}^*_M (t, -,-) \}$

5. If L = L(M) for some STM M

   ==> L is said to be *recursively enumerable (r.e.)*

6. If L = L(M) for some total *STM M*

   ==> L is said to be *recursive*

7. If $\sim L =_{def} \Sigma^* - L = L(M)$ for some STM M (or total STM M)

   ==> L is said to be *Co-r.e. (or Co-recursive, respectively)*

Some examples

Ex1: Find a STM to accept $L_1 = \{ w \# w \mid w \in \{a,b\}^* \}$

note: L_1 is not CFL.

The STM has tape alphabet $\Gamma = \{a, b, \#, -, □, [\}$ and behaves as follows: on input $z = w \# w \in \{a,b,\#\}^*$

```
1.  if z is not of the form $\{a,b\}^* \# \{a,b\}^*$ => goto reject
2.  move left until `[` is encountered and  in that case move right
3. while I/P = '-' move right;
4. if I/P = 'a' then
     4.1  write '-'; move right until # is encountered; Move right;
     4.2  while I/P = '-' move right
     4.3  case (I/P) of {  'a' : (write '-'; goto 2);     o/w: goto reject

5. if I/p = 'b' then … // like 4.1~ 4.3
6. If I/P = '#' then      // All symbols left to # have been compared
   6.1 move right
     6.2  while I/P = '-" move right
```

More detail of the STM

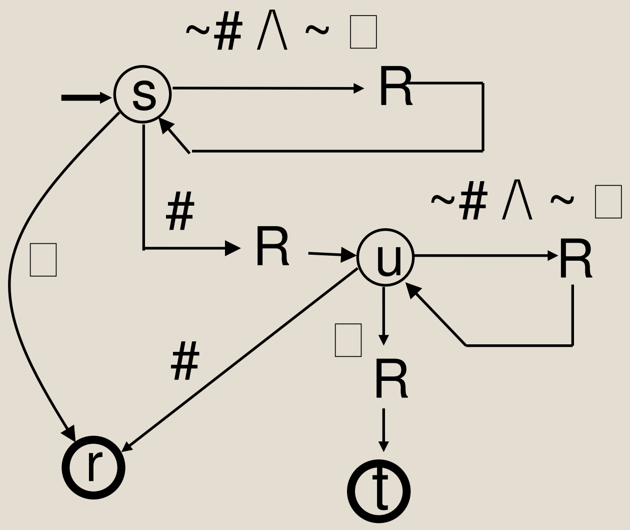
Step 1 can be accomplished as follows:

```
1.1  while (~# /\ ~ □)    R; // or equivalently, while (a \/ b\/[) R
        if □ => reject    // no # found on the input
        if # => R;
1.2  While ( ~# /\ ~ □ ) R;
        if □ => goto accept [or goto 2 if regarded as a subroutine]
        if #    => goto Reject;      // more than one #s found
```

Step 1 requires only two states:

Graphical representation of a TM

•



$$\begin{tikzcd} p \arrow[r, "\text{cnd}", "\text{ACs}"'] & q \end{tikzcd}$$

means:

if (state = p) /\ (cnd true for i/p)

then 1. perform ACs and 2. go to q

ACs can be primitive ones: R, L, a,…

or another subroutine TM M_1.

Ex: the arc from s to s implies the existence of 4 instructions:

(s, a, s, R), (s,b,s,R), (s, [,s,R), and (s,-, s,R)

Tabular form of a STM

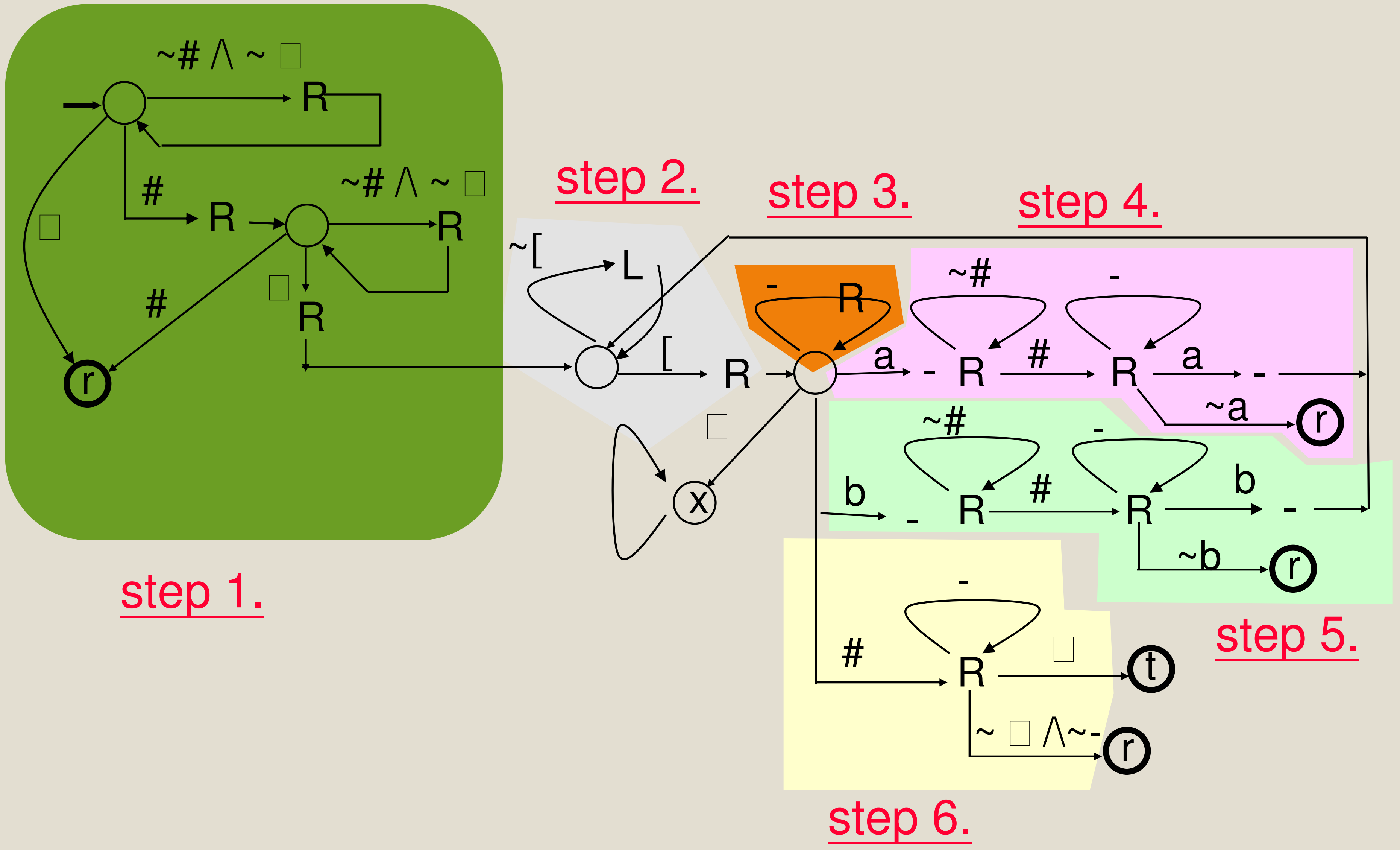
- Translation of the graphical form to tabular form of a STM

δ Γ Q	[	a	b	#	-	□
>s	s,R	s,R	s,R	u,R	x	r,x
u	x	u,R	u,R	r,x	x	t, □
tF	halt	halt	halt	halt	halt	halt
rF	halt	halt	halt	halt	halt	halt

X means don't care

The rows for t & r indeed need not be listed!!

The complete STM accepting L_1



R.e. and recursive languages

Recall the following definitions:

1. M is said to *halt* on input x if either M accepts x or rejects x.
2. M is said to *loop* on x if it does not halt on x.
3. A TM is said to be *total* if it halts on all inputs.
4. The language accepted by a TM M,

   $L(M) =_{def} \{x \in \Sigma^* \mid$ x is accepted by M, i.e., $(s, [x \,\square\, ^{\omega} ,0) \mid\!\!-\!\!-^*_{M} (t, -,-) \}$

5. If L = L(M) for some STM M

   ==> L is said to be *recursively enumerable (r.e.)*

6. If L = L(M) for some total *STM M*

   ==> L is said to be *recursive*

7. If $\sim L =_{def} \Sigma^* - L = L(M)$ for some STM M (or total STM M)

   ==> L is said to be *Co-r.e. (or Co-recursive, respectively)*

Recursive languages are closed under complement

Theorem 1: Recursive languages are closed under complement (i.e., If L is recursive, then $\sim L = \Sigma^* - L$ is recursive.)

pf: Suppose L is recursive. Then L = L(M) for some total TM M.

Now let M* be the machine M with accept and reject states switched.

Now for any input x,

- $x \notin \sim L \Rightarrow x \in L(M) \Rightarrow icfg_M(x) \mid\!-_M^* (t,-,-) \Rightarrow$
- $icfg_{M*}(x) \mid\!-_{M*}^* (r^*,-,-) \Rightarrow x \notin L(M^*)$.
- $x \in \sim L \Rightarrow x \notin L(M) \Rightarrow icfg_M(x) \mid\!-_M^* (r,-,-) \Rightarrow$
- $icfg_{M*}(x) \mid\!-_{M*}^* (t^*,-,-) \Rightarrow x \in L(M^*)$.

Hence $\sim L = L(M^*)$ and is recursive.

Note. The same argument cannot be applied to r.e. languages. (why?)

Exercise: Are recursive sets closed under union, intersection, concatenation and/or Kleene's operation ?

Some more terminology

Set : Recursive and recursively enumerable(r.e.)

predicate: Decidability and semidecidability

Problem: Solvability and semisolvabilty

- P : a statement about strings (or a property of strings)
- A: a set of strings
- Q : a (decision) Problem.

We say that

1. P is decidable <==> { x | P(x) is true } is recursive
2. A is recursive <==> "$x \in A$" is decidable.
3. P is semidecidable <==> { x | P(x) is true } is r.e.
4. A is r.e. <==> "$x \in A$" is semidecidable.
5. Q is solvable <=> Rep(Q) $=_{def}$ {"P" | P is a positive instanc of Q } is recursive.
6. Q is semisolvale <==> Rep(Q) is r.e..