THEORY OF FORMAL LANGUAGES · LECTURE 10

Sequential Operations of Turing Machines

Chaining simple Turing machines into macros — move, find, erase, copy, translate, and add — then composing macros into complete algorithms.

Prof. K. R. Chowdhary

Professor of CS

ROADMAP

What We'll Cover

1. **Sequential operation of Turing machines**
   How the output tape of one TM becomes the input tape of the next — chaining Z then S.

2. **Macros: TMs as reusable building blocks**
   The same idea as assembly or C macros — a named, self-contained unit of tape-editing behavior.

3. **Seven core macros**
   MR, ML, FR, FL, Erase, Copy, Translate, and Add — the toolkit every later construction is built from.

4. **Composing macros into algorithms**
   Two worked examples: swapping two numbers, and multiplying a number by 2.

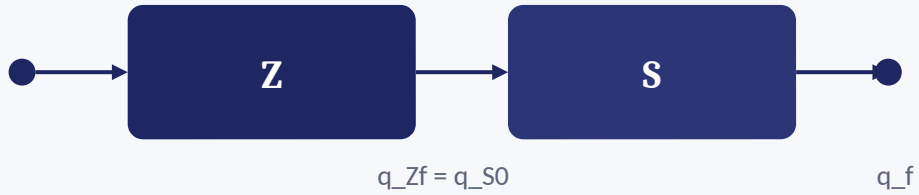
Chaining Turing Machines

A single Turing machine is designed for one task. To perform something more complex, several machines can run in sequence: the tape left behind by one machine becomes the starting tape for the next.

WORKED IDEA — Z, THEN S

Run Z first — the tape is rewritten to the code for 0.
Run S next, on Z's output — the code for 0 becomes the code for 1.
Z's final state becomes S's initial state.

SYMBOLIC VIEW (FIG. 10.2)



IN PLAIN ENGLISH

Think of it as a pipeline: each machine's halting configuration is simply handed to the next machine as its starting configuration. Nothing is shared except that one tape. If the input isn't in the format the next machine expects, you get garbage in, garbage out — so the tape format matters as much as the machines themselves.

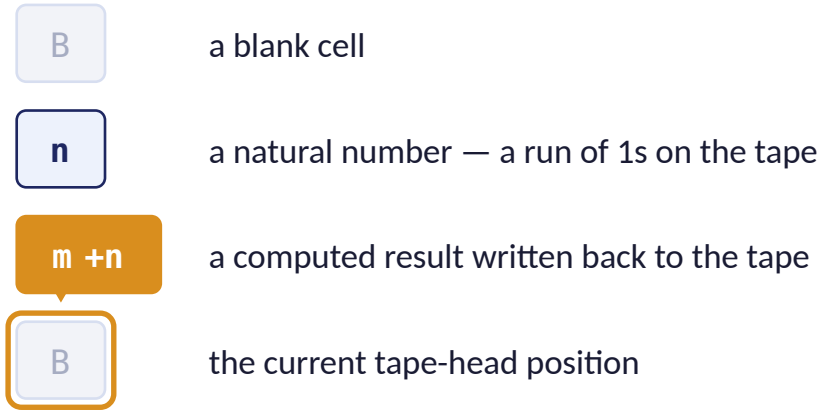
WHY THIS MATTERS

Z and S can now be treated as components of larger machines — the same way assembly or C code is built from reusable macros.
That's exactly where we're headed next.

Turing Machines as Macros

Component machines like Z and S are called macros — the same concept used for reusable, named routines in assembly and C. The rest of this lecture catalogs the most common macros, then shows how to compose them.

READING THE TAPE DIAGRAMS ON THE NEXT SLIDES



i **IN PLAIN ENGLISH**

A macro's contract is: given the tape in a specific starting shape, it halts with the tape in a specific ending shape — and it never touches anything outside the region it's documented to affect. That contract is all you need to know to use a macro as a building block; you don't need to know its internal states at all.

MR_i (Move Right) Macro

MACRO

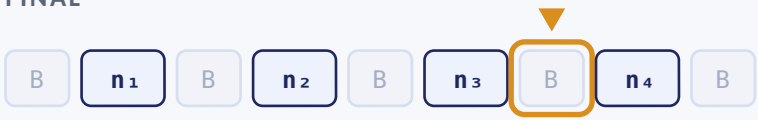
$MR_i(x)$ — moves the tape head right, past i consecutive numbers

Advances the tape head to the right, skipping over i whole numbers, so it lands on the blank just before the (i+1)-th number. Illustrated here for MR_2 (Fig. 10.3), advancing past two arguments to reach the start of the 3rd.

TAPE CONFIGURATION

INITIAL

FINAL



MR_2 shown: the head starts parked on n_2 and ends parked just before n_4, having stepped cleanly over two whole numbers.

IN PLAIN ENGLISH

A move macro never touches tape contents to the left of where it started. It needs at least i numbers to its right to be well-formed input — give it less, and the behavior is undefined.

ML_i (Move Left) Macro

MACRO

$ML_i(x)$ — moves the tape head left, past i consecutive numbers

The mirror image of MR_i: the head steps backward over i numbers instead of forward. Everything to the right of the starting position is left completely untouched.

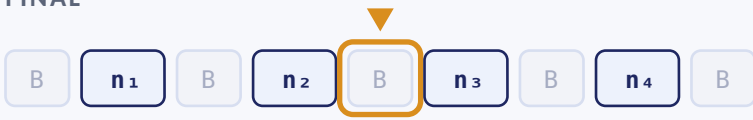
IN PLAIN ENGLISH

MR and ML are pure repositioning — no rewriting. Every more elaborate macro ahead is built by combining moves like these with reads and writes.

TAPE CONFIGURATION

INITIAL

FINAL

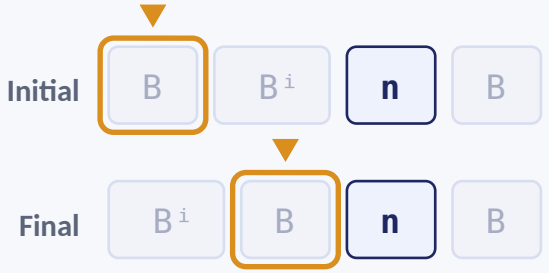


ML_2 shown: the head starts past all four numbers and ends parked between n_2 and n_3, having stepped back over two whole numbers.

FR and FL (Find Right / Find Left) Macros

FR moves the head to the first number it can find to the right; FL does the same to the left. Both stop one cell short of the number — parked on the last blank before it — rather than landing on the number itself.

FR — FIND RIGHT



i ≥ 0 blanks separate the head from the number. FR crosses them and stops one cell short of it.

FL — FIND LEFT

Same idea, mirrored: the head crosses i blanks leftward and again stops one cell short of the number.

IN PLAIN ENGLISH

“Find” macros are how later constructions locate data without knowing in advance how far away it is — they search, rather than moving a fixed distance like MR/ML do.

E_k (Erase) Macro

MACRO

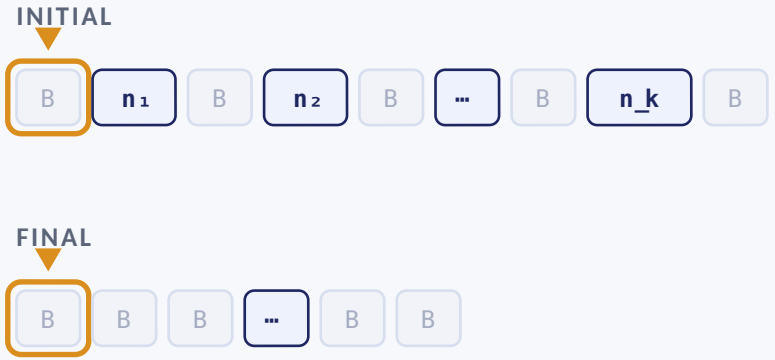
$E_k(x)$ — erases k consecutive numbers

Wipes out a run of k numbers, turning every cell in that span into a blank, then halts with the head back where it started. Nothing before or after the erased span is affected.

IN PLAIN ENGLISH

Erasing doesn't shrink the tape or shift anything leftward — the numbers simply become blanks in place, k ≥ 1. This matters when a later macro needs the original spacing preserved.

TAPE CONFIGURATION



The whole erased region collapses to blanks of the same physical extent — nothing shifts to fill the gap.

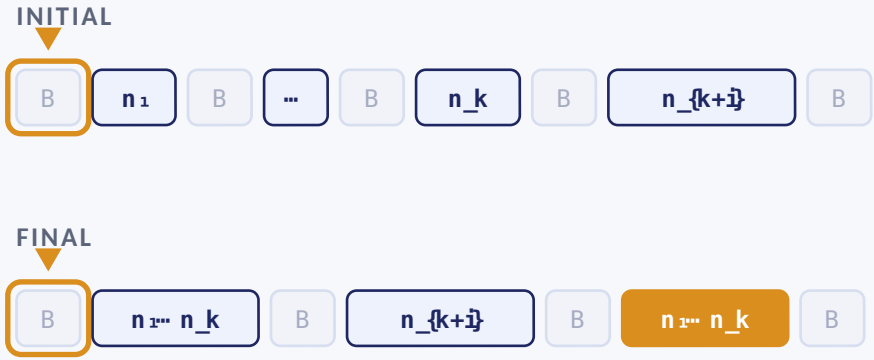
$CPY_{k,i}$ (Copy) Macro

MACRO

$CPY_{k,i}(x)$ — copies the first k of (k+i) numbers to the end

Expects k+i numbers on the tape, with blank space assumed to follow. It duplicates the first k of them and appends the copy after the (k+i)-th number, so the final tape holds 2k+i numbers in total.

TAPE CONFIGURATION



The head returns to its starting position; the newly appended copy (highlighted) is the macro's only output.

IN PLAIN ENGLISH

$CPY_{3,2}$ copies the first 3 numbers to the 5th position, giving 8 total. $CPY_{3,0}$ copies them right after the 3rd number, giving 6 total — i is just how many extra numbers sit in between.

T (Translate) Macro

MACRO

T(x) — relocates the first number to just right of the head

Moves a distant number so it sits immediately to the right of the tape head, then returns the head to exactly where it started. Unlike FR, the head doesn't travel — the data comes to it.

TAPE CONFIGURATION

INITIAL

FINAL

The i blanks that used to separate the head from n are pushed after n instead — the head itself never leaves its start cell.

IN PLAIN ENGLISH

Compare with FR, which has the same initial shape: FR moves the head to the number, while T moves the number to the head instead.

A (Add) Macro

MACRO

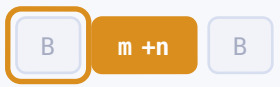
A(x) — adds two adjacent numbers, replacing them with their sum

Reads the two numbers immediately to the right of the head and collapses them into one number equal to their sum — the arithmetic building block every later construction (multiplication, and beyond) is assembled from.

TAPE CONFIGURATION

INITIAL

FINAL



Two separate numbers go in; a single combined number comes out, in the same span the two originals occupied.

IN PLAIN ENGLISH

This is the first macro that actually computes something rather than just rearranging the tape — everything before it was pure bookkeeping. Addition is where macros start doing arithmetic.

PUTTING IT TOGETHER

Composing Macros into Complete Algorithms

With MR, ML, FR, FL, Erase, Copy, Translate, and Add in hand, entire algorithms can be built by chaining macros exactly as Z and S were chained at the start of the lecture. Two examples follow: swapping two numbers, and multiplying a number by 2.

WORKED EXAMPLE

Example 10.1 — Swap Two Numbers

Construct a sequence of macros that exchanges two numbers m and n sitting on the tape.

i **IN PLAIN ENGLISH**

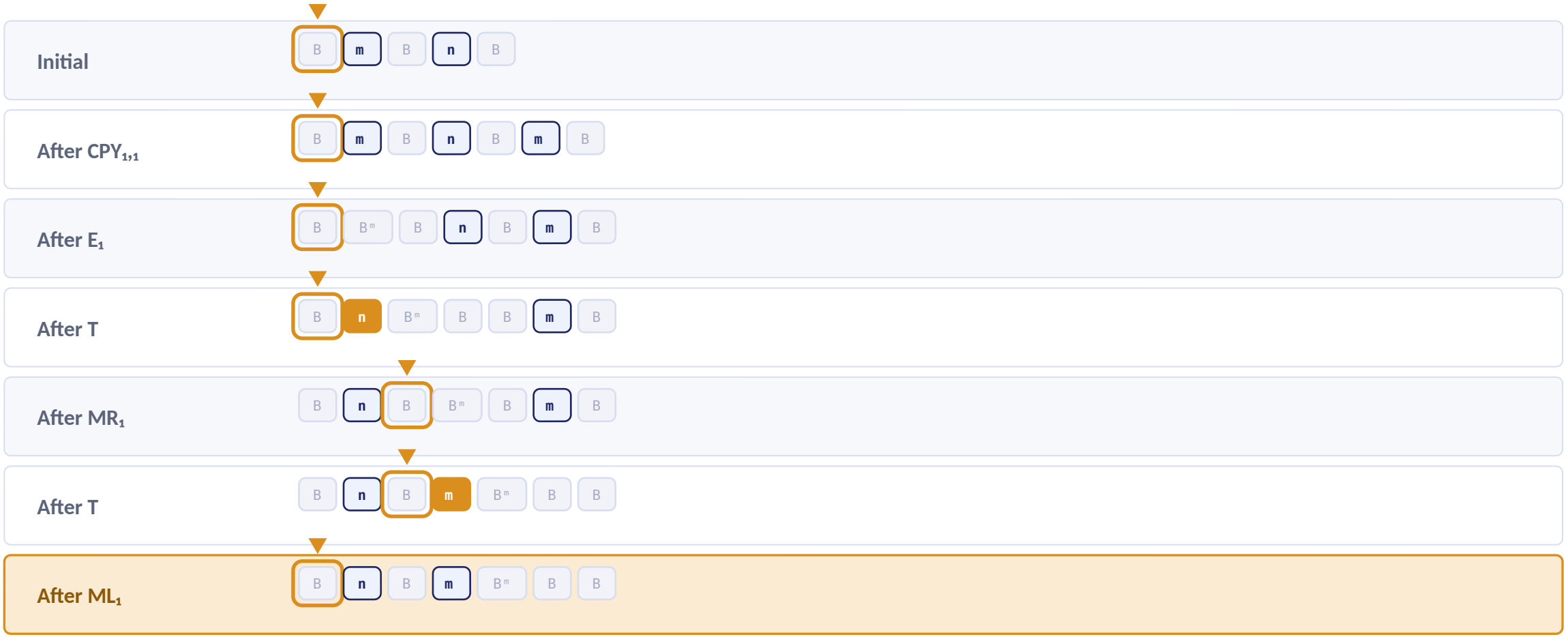
The strategy: make a spare copy of m, erase the original, shuffle n leftward into m's old place, then move the spare copy of m into n's old place. Six small, well-understood macros — none of them individually "knows" about swapping — combine into behavior none of them has alone.

SIX MACROS IN SEQUENCE (FIG. 10.4)

$\mathbf{CPY_{1,1}}$	make a spare copy of m at the end
$\mathbf{E_1}$	erase the original m
T	bring n next to the head
$\mathbf{MR_1}$	step past n
T	bring the spare copy of m forward
$\mathbf{ML_1}$	step back to park before n

The full step-by-step tape trace follows on the next slide.

Example 10.1 — Swap, Step by Step



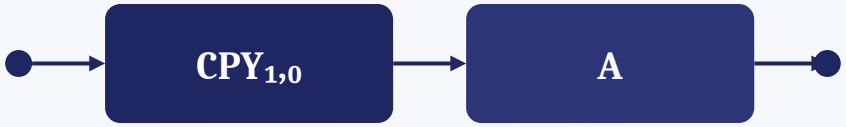
m and n have swapped places, and the head is parked on the blank just before n — ready for whatever comes next.

WORKED EXAMPLE

Example 10.2 — Multiply a Number by 2

Construct sequential macros that turn a number n on the tape into n + n.

TWO MACROS IN SEQUENCE (FIG. 10.5)



Copy n once, then add the copy to the original.

IN PLAIN ENGLISH

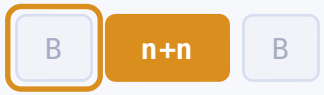
This is the simplest possible composition in the whole lecture — just two macros — but it already shows the pattern behind every arithmetic construction to come: duplicate the data you need, then let Add combine it. Multiplication by any constant k, later, is nothing more than k−1 copies chained the same way.

TAPE TRACE

INITIAL

AFTER $CPY_{1,0}$

AFTER A (FINAL)



Two macros, zero new machinery — doubling falls straight out of copy + add.

SUMMARY

Key Takeaways

1

Sequencing is just tape hand-off

One machine's halting configuration becomes the next machine's starting configuration — no shared state beyond that.

2

Macros hide the machine states

You only need a macro's initial → final tape contract to use it — exactly like calling a function without reading its source.

3

Seven macros cover most needs

MR, ML, FR, FL, Erase, Copy, and Translate handle bookkeeping; Add is the first macro that actually computes.

4

Complex behavior from simple parts

Swap needed six macros and Add falls out of just two — neither behavior exists in any single macro alone.

Thank You

Questions on sequential composition, any of the macros, or the worked examples?

Prof. K. R. Chowdhary

Professor of CS · Theory of Formal Languages, Lecture 10