

CS 3313

Foundations of Computing:

**Computing Functions on a
Turing Machine**

<http://gw-cs3313-2021.github.io>

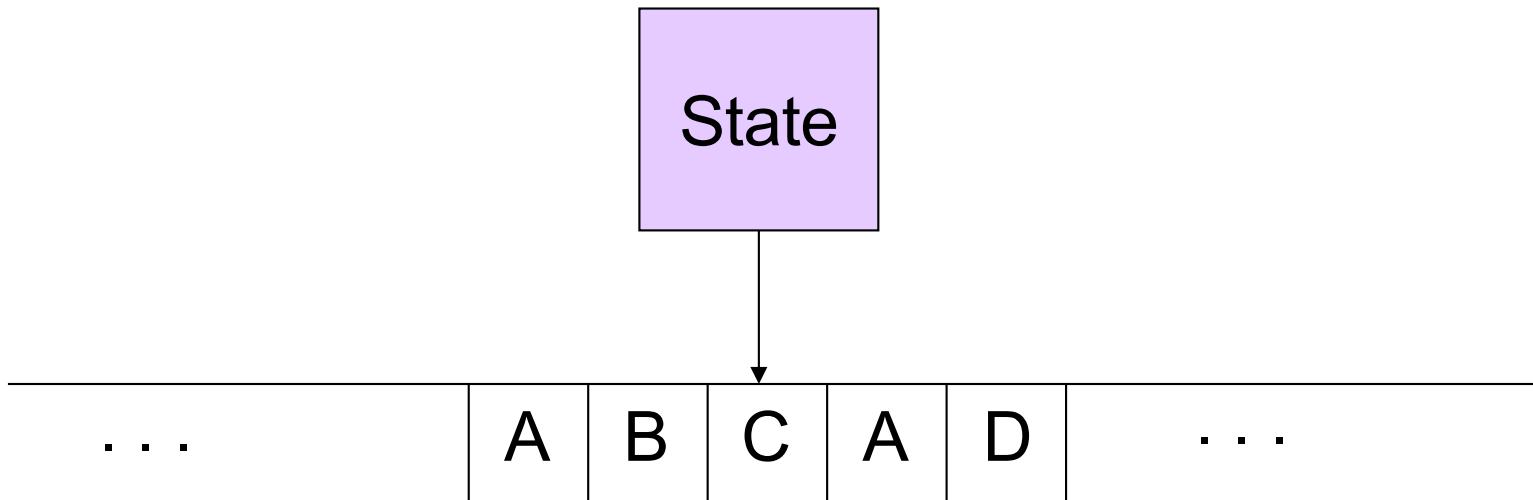
Next..

- Turing Machine model
 - TM as an automaton
- Today: Computing functions on a Turing machine
 - Turing machine as a “computer”
 - First step is encoding the integer arguments to the TM
- Next....TM “programming” techniques
 - Storage in the state (you’ve seen this) – ‘caching’ a value
 - Checking symbols
 - Shifting over (skipping) tape symbols
 - Subroutines...!!!
 - Multiple tracks on the tape

Turing Machine

Action: based on the (i) state and (ii) the tape symbol under the read/write head:

- (1) change state, (2) write a symbol back to the tape and (3) move the head (left or right) one location/square on the tape.



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

Turing-Machine Formalism

- A TM is described by:
 1. A finite set of *states* Q .
 2. An *input alphabet* Σ .
 3. A *tape alphabet* Γ (contains Σ).
 4. A *transition function* $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$
 5. A *start state* q_0 (in Q).
 6. A *blank symbol* B (or \square) in $\Gamma - \Sigma$
 - All tape except for the input is blank initially.
 7. A set of *final states* $F \subseteq Q$

The Transition Function

- Takes two arguments:
 1. A state, in Q .
 2. 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 – move the tape head to the Left or Right
- Convention: If undefined then TM halts
 - If it halts in a final state then it accepts
 - If it halts in a non-final state then it rejects

Instantaneous Descriptions (ID) of a Turing Machine

- Initially, a TM has a tape consisting of a string of input symbols surrounded by an infinity of blanks in both directions.
- The TM is in the start state, and the head is at the leftmost input symbol.
- An ID is a string $\alpha q \beta$, where $\alpha \beta$ includes the tape between the leftmost and rightmost nonblanks.
 - The state q is immediately to the left of the tape symbol scanned
 - If q is at the right end, it is scanning B.
 - If q is scanning a B at the left end, then consecutive B's at and to the right of q are part of α .

Formal Definition of Moves: Instantaneous Description (ID)

- At any instant in time, the TM is in a state q , its tape head is reading some symbol Z , the string α is to the left of the tape head, and the string β is to the right of the tape head:

this ID is denoted as $\alpha q Z \beta$

- Moves of a TM take TM from one ID to another..be defined using $ID_1 \vdash ID_2$ and $ID_1 \vdash^* ID_2$ to represent “in one move” and “in zero or more moves,”
- If $\delta(q, Z) = (p, Y, R)$, then $\alpha q Z \beta \vdash \alpha Y p \beta$
 - ◆ If Z is the blank 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, $q Z \beta \vdash p B Y \beta$

Languages of a TM

- A TM defines a language by final state, as usual.
- $L(M) = \{w \mid q_0 w \vdash^* I, \text{ where } I \text{ is an ID with a final state}\}$.
 - TM halts in this configuration
 - Alternate definition: accepts as long as state is a final state
- A language that is accepted by a TM is called a *recursively enumerable language*
 - Strings that can be enumerated by a TM
 - TM may not halt on an input not in the language
- A language that is accepted by a TM that halts on all inputs is called a *recursive language*

Turing Machine as a transducer...i.e., as a computer of functions

- TM M starts with input string w and in ID q_0w
- Let TM halts (in accepting) state q_f and y is string on the tape
 - $q_0w \vdash_M^* q_fy$
- *Then*, the **function computed by the TM M is $y=f(w)$**

Turing Machine as a computer of functions: Encoding integers

- The simplest scheme to encode an integer uses the Unary representation
 - One symbol a , and integer $n \geq 0$ is represented as a^n
 - Unary representation is what we start with....
- Our notation: We will use symbol 0 to as the unary symbol
 - Integer $n > 0$ encoded as 0^n
 - Note: could pick any other symbol...1, 2,
 - Textbook uses 1 instead of 0

- Given an integer n , a TM computers $f(n)$ if

$$q_0 0^n \vdash_M^* q_f 0^{f(n)} \text{ where } q_f \text{ is a final state.}$$

- Ex: $q_0 0^n \vdash_M^* q_f 0^{2n}$ computes $f(x) = 2x$

Turing Machine as a computer of functions: Encoding integers

- What if we have multiple arguments...each encoded in unary ?
 - Need another symbol to separate the arguments
- Notation: we use 1 to separate arguments
 - Textbook uses 0 (it swaps the roles of the 0's and 1's)
- To specify two arguments x, y we place $0^x 1 0^y$ on the tape.
- Given inputs x, y a TM M computes $f(x, y)$ if
$$q_0 0^x 1 0^y \vdash_M^* q_f 0^{f(x,y)} \text{ where } q_f \text{ is a final state.}$$
 - Ex: $q_0 0^x 1 0^y \vdash_M^* q_f 0^{x+y}$ computes $f(x, y) = x+y$
- **Generalize to any n arguments:** $q_0 0^{x_1} 1 0^{x_2} 1 \dots 1 0^{x_n} \vdash_M^* q_f 0^{f(x_1, x_2, \dots, x_n)}$
- What about multiple outputs ? Again, place separators (use different separator such as 11 to distinguish?)

$$q_0 0^x 1 0^y \vdash_M^* q_f 0^{f(x,y)} 11 0^{g(x,y)}$$

Definition: Functions

- A function $f(x_1, x_2, \dots, x_n)$ computed by a Turing machine is called a *partial recursive function*
 - Analogous to Recursively Enumerable (r.e.) languages
 - TM may not halt on all inputs
- If a function $f(x_1, x_2, \dots, x_n)$ is defined for all values of x_1, x_2, \dots, x_n and computed by a Turing machine, then we say that f is a *total recursive function*
 - Analogous to recursive languages
 - TM halts on all inputs since function value is defined for all inputs
- The terms recursive and partial recursive functions were introduced independent of TMs.

Example 1

- Representation Model: (a) blanks are ignored or (b) no blanks in the middle of an output value
 - We will use the “no blank tape symbols in the middle of an output value”
- $f(x,y) = x+y$
 - $0^x 1 0^y \vdash^* 0^{x+y}$
 - Algorithm: Copy every 0 to the left of 1 to the right end.
 - Copy 0^x to right after 0^y
 - Erase the 1, and halt
 - $0^x 1 0^y \vdash^* 0^{x-1} 1 0^{y+1} \vdash^* 1 0^{x+y} \vdash^* 0^{x+y}$

Example 2: $f(x,y) = x - y$

- Ex 2: $f(x,y) = x - y$ assume for simplicity that $x \geq y$
- Algorithm: Input is $0^x 1 0^y$ (tape head is at left end)
 - For every 0 left of the separator 1, move right and erase one zero on right of 1
 - Until no 0's to the right of 1
- 1. Replace 0 with B, “skip” right over 0's until you read a 1
- 2. Skip right over 0's until you see a B, move left
 - If all 0's from 0^y have been erased, then we will be reading a B immediately to the right of 1. If this is the case, then we need to erase the separator 1 and we are done.
- 3. Replace 0 with a B (i.e., erase one 0 from 0^y) go left
- 4. Skip right over 0,1 until we hit left end of 0^x – i.e., until we read B, goto 1.

Example 3: $f(x,y) = x*y$ (Multiplication)

- $f(x,y) = x*y$ assume for simplicity that $x,y \geq 1$
- $q_0 0^x 1 0^y \vdash^* q_f 0^{x*y}$
- We have TMs (programs) to (1) ADD and (2) SUBTRACT
- How do we write a program to multiply two integers, assuming we have a program to ADD two integers ?
 - Multiplication through repeated addition

`i = x`

`while i > 0`

`i = i - 1`

`mult = ADD(mult, y) /* subroutine call !! */`

Example 3: $f(x,y) = x*y$ (Multiplication):

Notations and Specifications

- Input is $0^x 1 0^y$ and output should be 0^{x*y}
- Unlike addition and subtraction, we need to keep 0^{x-i} on the tape to keep track of value of i , and we need to keep 0^y on tape so we can keep “adding” to the running sum Mult
- Therefore introduce new output field – how ?
 - Add a separator at end of input and write the value of Mult after that.
 - $0^x 1 0^y 1 0^{mult}$
- $q_0 0^x 1 0^y \vdash^* 0^x 1 0^y 1 \vdash^* 0^{x-i} 1 0^y 1 0^{i*y} \vdash^* 1 0^y 1 0^{x*y} \vdash^* q_f 0^{x*y}$

Example 3: $f(x,y) = x*y$ (Multiplication): specifications of the ADD subroutine

- Key to the Multiplication is repeated calls to the “ADD”
 - First step is specification of this “subroutine”
- This subroutine essentially copies n 0's from left of 1 to the right of 1
- Specifications: $q_1 0^n 1 0^k \vdash_{\text{ADD}}^* q_n 0^y 1 0^{k+n}$
- Algorithm: for every 0 to the left of the 1, copy to the right end (write 0);
repeat until no more 0's to the left
 - Similar to the ADD TM that we designed but.....
- Important: we want to call this ADD subroutine (to add y to running sum) repeatedly...so don't “lose” (erase) the 0^n
 - \Rightarrow *at the end, after copying the n 0's we should be able to restore 0^n*
 - \Rightarrow *instead of erasing the 0, write a new symbol 2 and then when done restore 2's to 0's*

Example 3: $f(x,y) = x*y$ (Multiplication): specifications of the ADD subroutine

1. Read 0, change to 2, move right (to copy the 0) IF you read 1 then goto 4
 - If you read a 1 then it means you are done copying the n 0's
2. Skip right over 0's and 1 until the first B: change B to 0, move left
3. Skip left over 0/1 until you read a 2 then go right and goto 1.
 - i.e., searching for the rightmost "checked" 0 in 0^n
4. (Restore 0^n) Change all 2's to 0's – keep going left changing 2's to 0's until you see a 1 (the left of 1 is another argument 0^x)

Example 3: $f(x,y) = x*y$ (Multiplication): specifications of the ADD subroutine

- Transition Function for ADD
 - Entry state = q_1
 - End state = q_5

Example 3: $f(x,y) = x*y$ (Multiplication)

- We have subroutine for ADD...we need to call it repeatedly

$i = x$

while $i > 0$

$i = i - 1$

$mult = ADD(mult, y)$ /* subroutine call !! */

$$q_0 0^x 1 0^y \vdash^* 0^x 1 q_1 0^y 1 \vdash_{ADD}^* 0^{x-i} 1 q_5 0^y 1 0^{i*y} \vdash^* q_f 0^{x*y}$$

To call ADD, go to state q_1

When ADD returns it is in state q_5

Example 3: $f(x,y) = x*y$ (Multiplication)

- Calling and returning from Subroutine

Example 3: $f(x,y) = x*y$ (Multiplication)

$$q_0 0^x 1 0^y \vdash^* 0^x 1 q_1 0^y 1 \vdash_{ADD}^* 0^{x-i} 1 q_5 0^y 1 0^{i*y} \vdash^* q_f 0^{x*y}$$

1. First (erase 0, ie., $x-1$) and go all the way to the end of input and add 1 and move left. /*this creates space for the output */
2. Go left to a 1, move right and call ADD
3. (return from ADD) Go left (to end of 0^{x-i})
 - Skip 0/1 until read B move right
4. Read 0, Erase 0, move right;
 - IF read 1 (then $i=0$) write B and goto 6.
1. Skip over 0's until read 1, move right and Call ADD (this returns in step 3)
2. (Done with Mult) so erase $10^y 1$
 - Read 0, Write B, go right until Read 1, Write B and accept.

Exercises

- 1: $f(n) = n^2$ $q_0 \theta^n \vdash^* q_f \theta^y$ where $y=n^2$
 - Think of what "pre-processing" needs to be done before calling the multiplication "function"
- 2: $f(n) = 2^n$
 - Input is θ^n and output should be θ^y where $y=2^n$