Languages
Grammars
Machines

Language

- Set of strings (words) over a finite alphabet.
- Examples:
  - {1, 111, 11111}
  - {11, 1111, 111111, …} Even length non-empty strings of ones.
  - $a^*b^* | a$ Regular expression.
  - {a, b, aa, bb ,aba, bab, aaa, bbb, …} Palindromes (not regular)
  - $a^n b^n$
  - $a^n b^n c^n$

Grammars

- Finite set of terminals (alphabet)
- Finite set of non-terminals (Variables that can be rewritten)
- Finite set of production rules. LHS can be rewritten by the RHS
- Start Symbol (by convention the non-terminal on the LHS of the first rule.)
- Example:

$$S \rightarrow aSa$$

$$S \rightarrow b$$

- Generates the language $a^n b a^n$

Grammars

- Left-Linear Grammar
- Context Free Grammar
- Context Sensitive Grammar

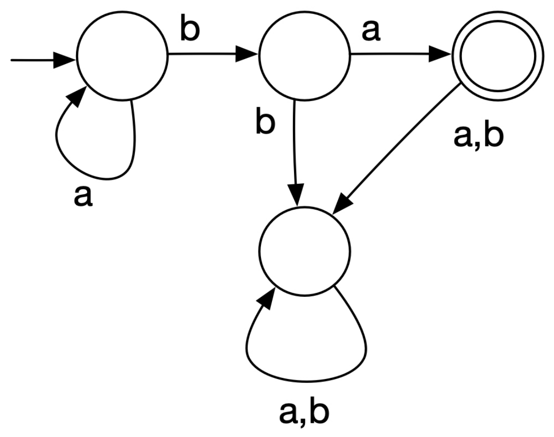
Machines

- Deterministic Finite Automata (DFA)
- NonDeterministic Finite Automata (NFA)
- Deterministic PushDown Automata
- PushDown Automata (PDA)
- Turing Machine

DFA

- Finite input alphabet
- Finite set of states
- Finite input
- Transition (State, input) → State (All possible are defined.)
- Start State
- Set of accepting states.
- Output is Yes/No
- Always in exactly one state. (Deterministic)

DFA

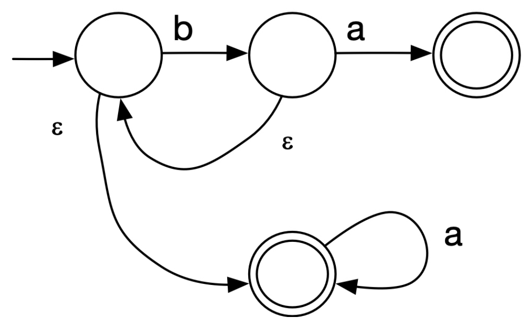


- Accepts the language: a^*ba

NFA

- Finite input alphabet
- Finite set of states
- Finite input
- Transition (State, input+empty) → Set of States
- Start State
- Set of accepting states.
- Output is Yes/No.
- As you run the machine, you can be in states simultaneously. Before you consume an input character, you add to the set of states you are in any state that you can get to using epsilon (empty) transitions. After all input is used, if the machine is in any accepting state, then you accept the word, otherwise reject.

NFA

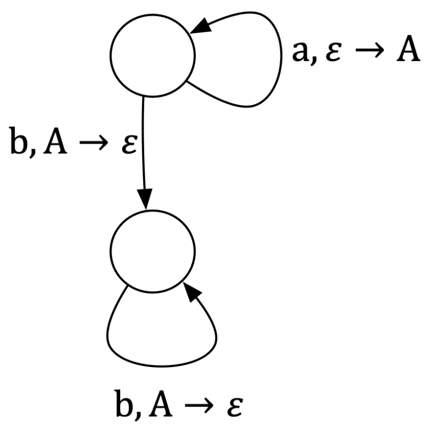


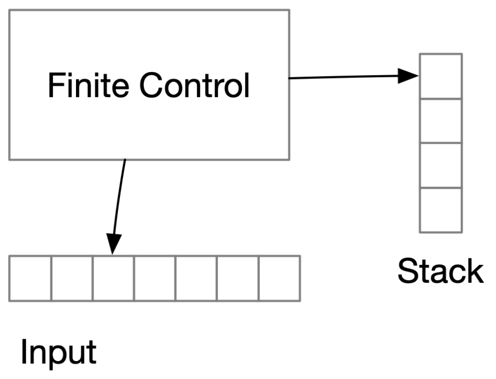
- Accepts the language: b^*a^*

PDA

- Finite input alphabet
- Finite stack alphabet
- Finite set of states
- Finite input
- Transition (State, input, current top) → (State, new top)
    - input is consumed, current top is popped, and new top is pushed.
- transition can have epsilon’s for
    - input (ignores it, input not consumed)
    - current top (ignores it, no pop)
    - new top (ignores it, no push)
- Start State
- Accept if input consumed and stack is empty (There are other conditions that can be used instead.)
- NonDeterministic (mulitple copies of the pda in different state/input/stack configurations.

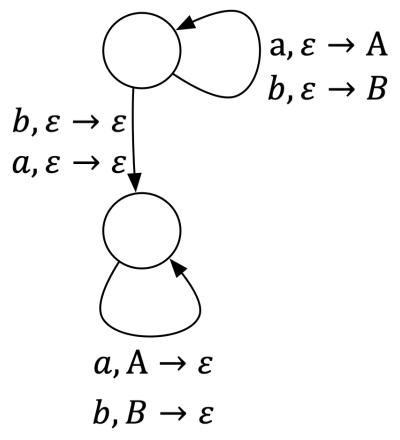
PDA

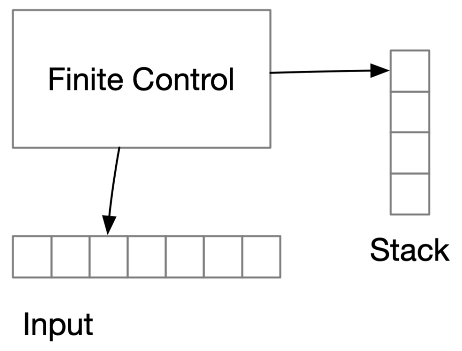




- Accepts the language: $a^n b^n$
- First state pushes an A for each a
- When we see b, we start matching and popping A's.
- If there are not enough b's, then we run out of input - No accept
- If there are too many b's, then we run out of A's on the stack - No accept
- This PDA is deterministic since we never have a choice for which transition to take.

PDA





- Accepts the language: Odd length palindromes over {a,b}
- First state pushes an A for each a and a B for each b it sees.
- At each step, we nondeterministically guess that we have seen the middle character.
- Second state matches off symbols on the stack with the remaining input.
- This PDA is non-deterministic since on any character we can either stay in the first state or transition to the second state.
- If we have an odd length palindrome, one of these guesses will be correct and the others will fail.

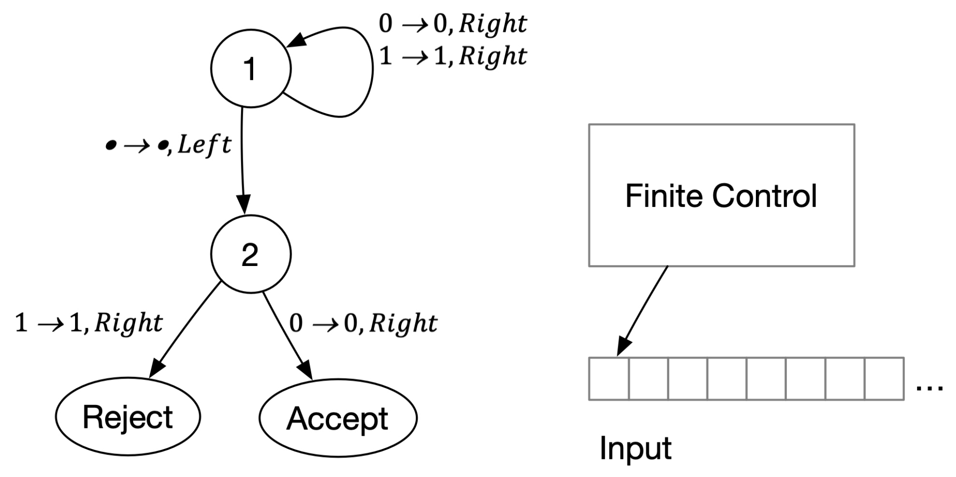
TM

- Finite tape alphabet
- Finite set of states
- A tape with an infinite number of cells.
- The input is stored at the front of the tape with all other cells blank.
- Transition (State, tape) →(State, new tape, {Left,Right})
- Usually deterministic (non-determinism doesn’t give you more power for TM)
- A state can be an accept or reject. If the TM transitions into one of these states, it will halt.
- Note: It is possible that a particular TM may never give you an answer.

TMs are messy

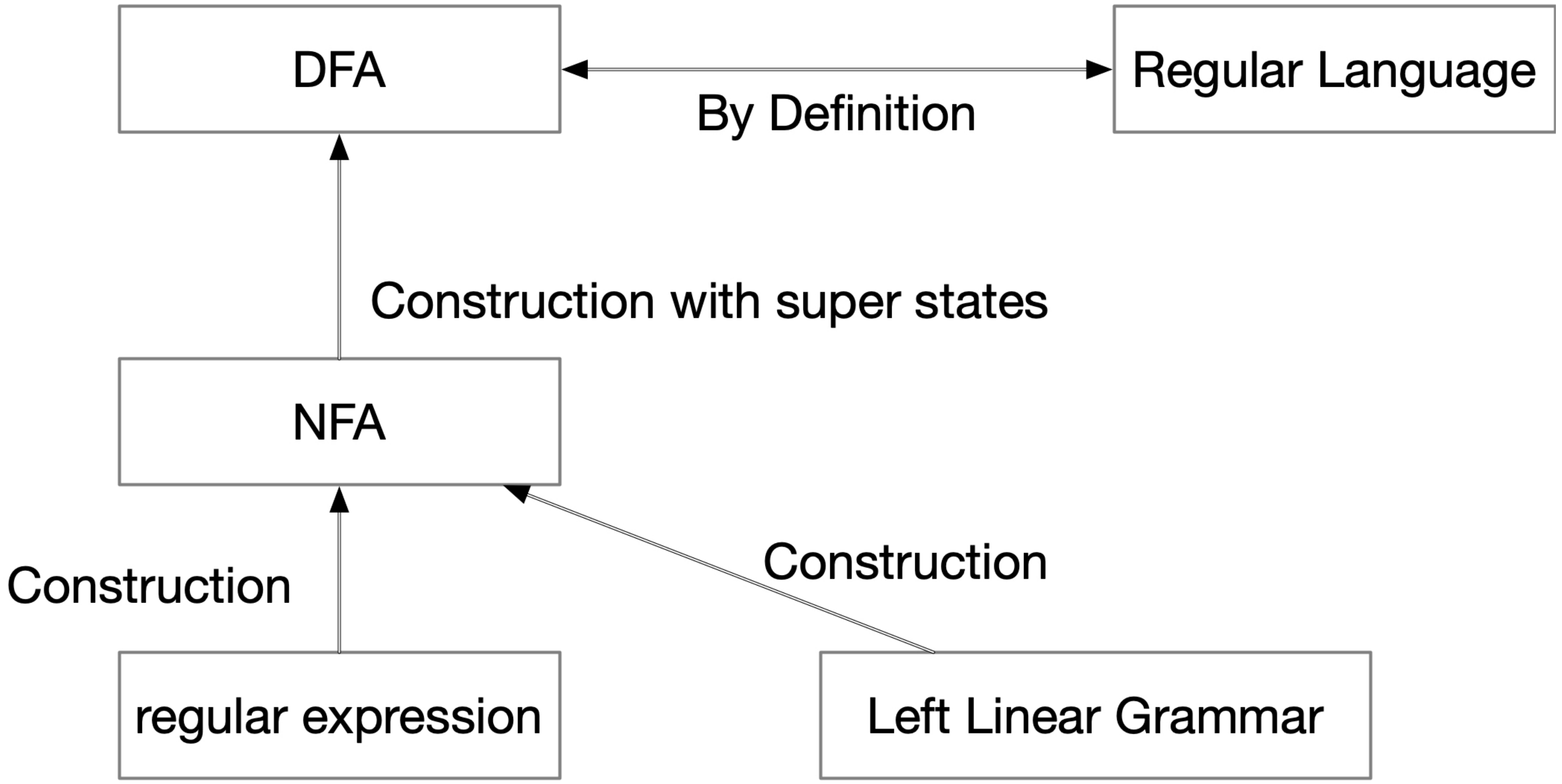
- In general, there are a number of things that you can not know about a TM. (You may be able to show it for some particular TM though.)
  - Does a TM halt on a given word.
  - Does a TM accept/reject a give word.
  - Does a TM accept a given language.
  - Does a TM reject all words.
  - Does a TM accept some word
  - Do two TMs accept the same language
  - Does the language accepted by a TM have some non-trivial property. (Non-trivial properties are those that true for every language.)

TM

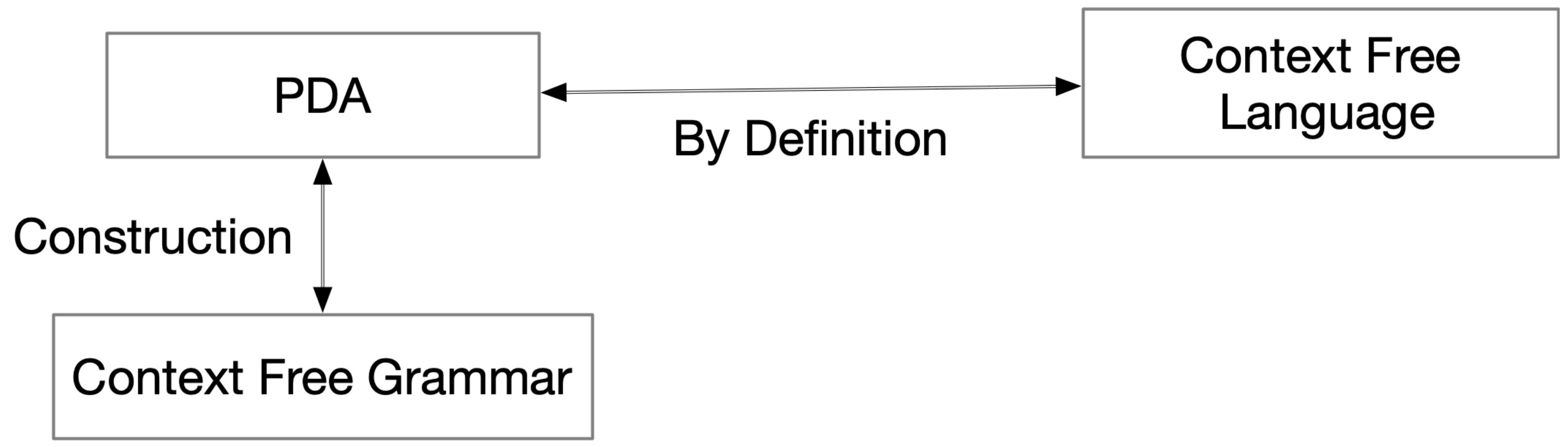


- Accepts the language: Even binary numbers.
- First state scans right until it find the end and then it moves the tape head to be over the last input symbol.
- If the last symbol is a zero, then the number is even and we accept.
- If the last symbol is a one, then the number is odd and we reject.
- This TM is pretty well behaved. For any non-empty input, it will halt.

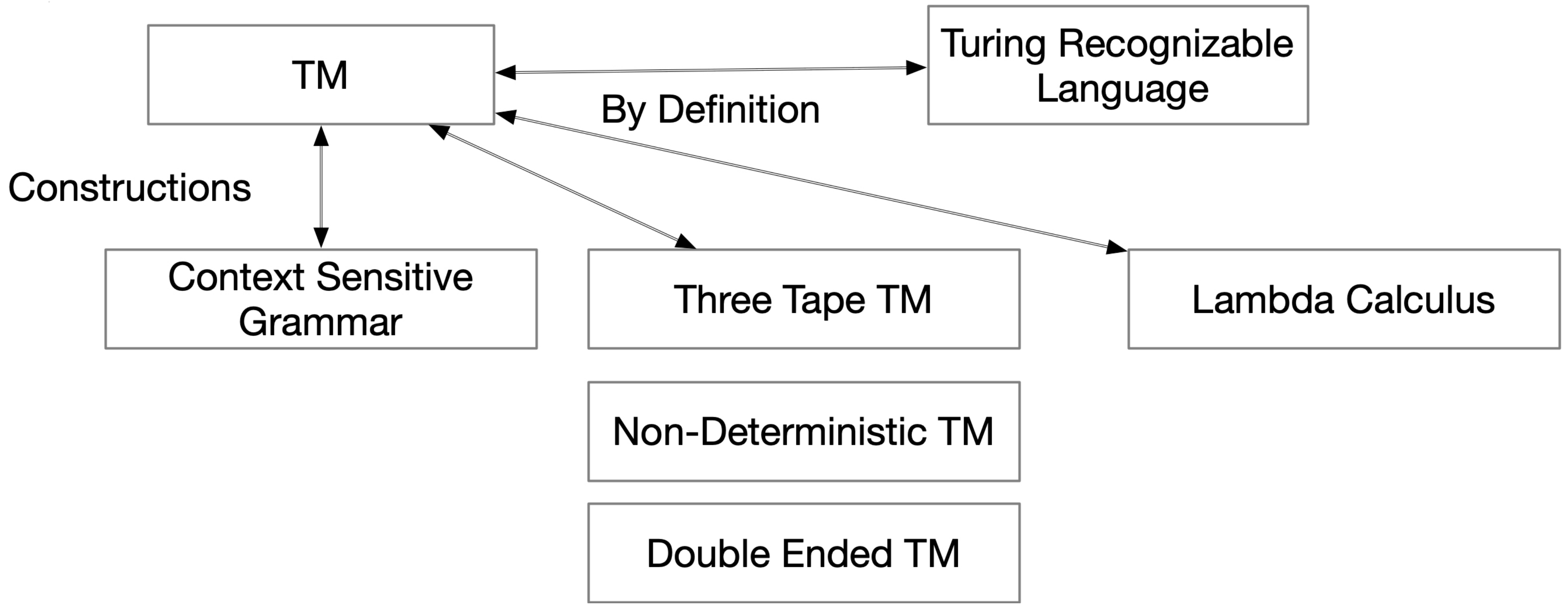
Constructions



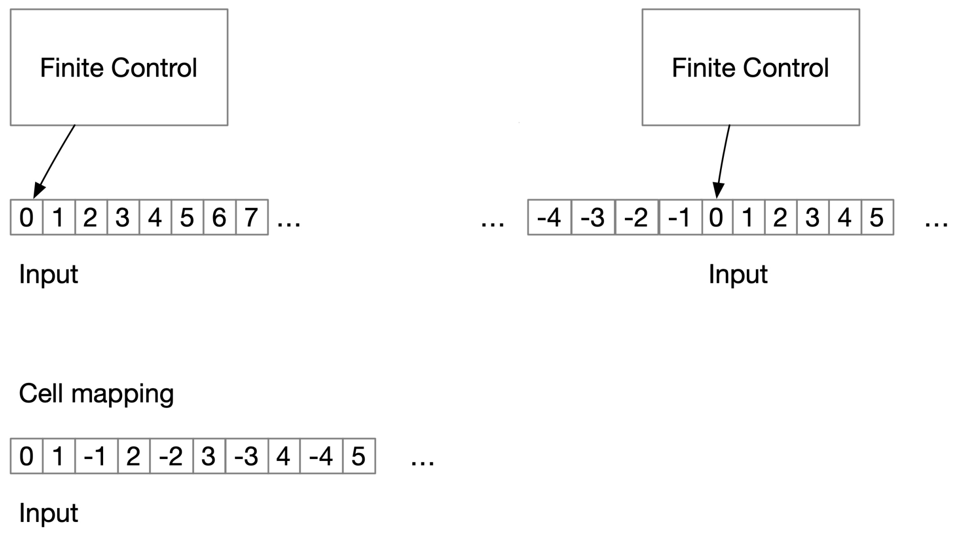
Constructions



Constructions

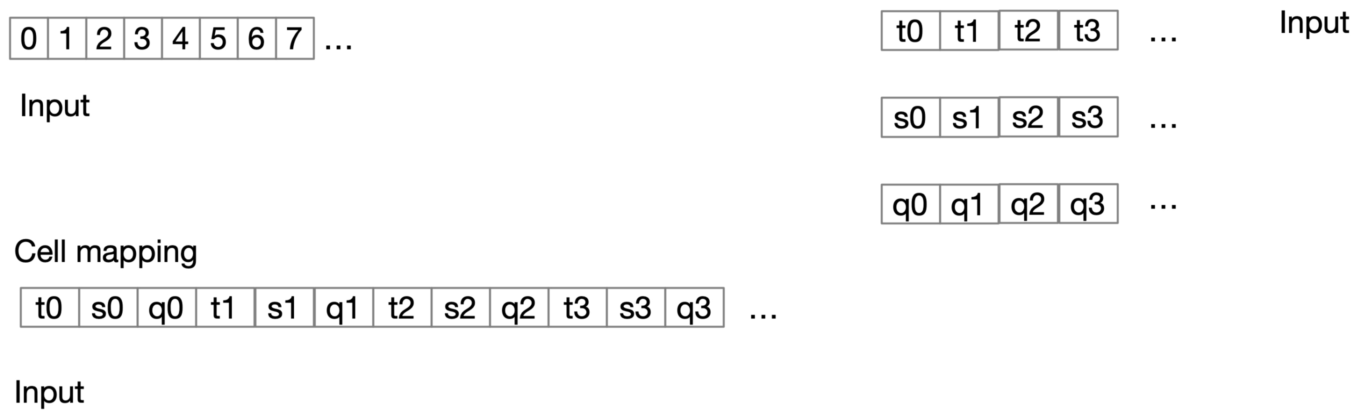


Double Ended to Single Ended



- To simulate a double ended TM, you need to show
  - How to map the squares.
  - How change the program. (E.G. Between every pair of states where there is a transition, add another state that skips over the cell from the other half of the tape.)

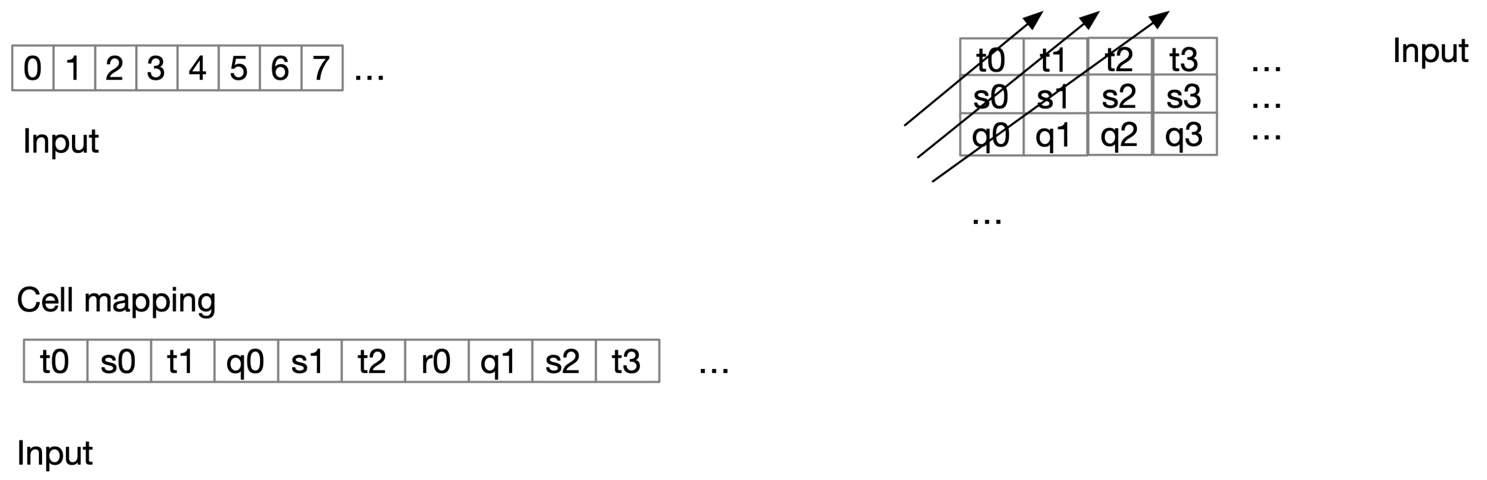
3 Tape to Single Ended



- One from each tape merge. (Cannot take all of tape 1 since that would use up all the spaces.)

- Generalizes to any finite number of tapes.

Grid
Single Ended



- Map up on the diagonals.
- Notice that we have infinitely many tapes!

Countably Infinite

- A set is Countably infinite if there is a one-to-one correspondence to the Natural numbers. These sets have a cardinality that is denoted as aleph null $\aleph_0$.
- Examples
  - Natural numbers
  - Odd natural numbers
  - Integers (Doubly Ended TM pattern)
  - Union of infinite sets (3-tape TM pattern)
  - Rational numbers (Grid TM pattern.)
  - Strings in a Language
  - Java Programs
- Not Countable
  - Real numbers
  - Languages

Rationals (positive)

	1	2	3	4	5	…
1	1/1	1/2	1/3	1/4	1/5	
2	2/1	2/2	2/3	2/4	2/5	
3	3/1	3/2	3/3	3/4	3/5	
4	4/1	4/2	4/3	4/4	4/5	
…						

- List them 1/1, 2/1, 1/2, 3/1, 2/2, 1/3, 4/1, 3/2, 2/3, 1/4, …
- Cross out the duplicates for the final mapping.

Finite Strings over some alphabet

- The set of all strings over an alphabet is countably infinite:
- Proof: Construct a way to list out all the strings. List every string in lexicographical order. Shorter strings come before longer strings.
- Example: Listing for bit strings
  - ϵ, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, …
  - Cannot just use the value of the bit string to order them as there are infinitely many representations of 0.
- Consequence: Any infinite subset of all strings over an alphabet is countably infinite!

Legal Programs

- Legal Programs are countably infinite
- There are infinitely many.
- We can convert any legal program into a bit string so are a subset of the bit strings.
  - Take the ASCII representation of the program.
  - Take the binary for the compiled program.

Languages are not Countable

- We will show this for languages over {a,b} using a diagonalization argument.
- Assume that they are countable. Then we can list them L1, L2, L3, …
- For each possible word it is either in the language (true) or not in the language (false)

	e	a	b	aa	ab	…
L1	T	F	F	F	F	
L2	T	T	T	T	F	
L3	F	F	F	F	F	
L4	T	F	T	F	T	
…						

- Construct a new Language L* that differs from Language Ln for the nth word.
- In the example L* would start F, F, T, T, ….
- No matter how we try to list them, we can always find a language that is missing. Therefore, the set of Languages is not countably infinite.
- They have the cardinality of $\aleph_1$.

Languages are not Countable

- There are more languages/functions than there are programs…
- There are orphan languages/functions (and uncountably infinite number of them) that are not computable. There is no program that will compute that function.
- We care about them… (reference the TM messy list)
- Closely related: There are more true formulas in predicate logic than there are proofs.
    - Goedel's incompleteness Theorem.
    - In any sufficiently complicated theory (definition of symbols and predicates) there are statements that are true but unprovable.
    - Sufficiently complicated means able to represent theorems in the theory and it doesn't take much. Addition over the natural numbers (Peano arithmetic) is enough.