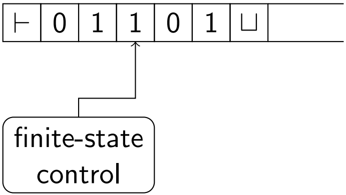
CS 475: Formal Models of Computation

Mahesh Viswanathan
vmahesh@illinois.edu
3232 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2023

Turing Machine



- A semi-infinite tape with $\vdash$ in leftmost cell
- Initially input stored on tape, with rest of the cell $\sqcup$
- In one step, machine reads symbol under head, and based on current state, changes state, writes a new symbol in cell, and moves head either L or R.

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,
- $\vdash \in \Gamma \setminus \Sigma$ is the *left endmarker*,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,
- $\vdash \in \Gamma \setminus \Sigma$ is the *left endmarker*,
- $\sqcup \in \Gamma \setminus \Sigma$ is the *blank symbol*,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,
- $\vdash \in \Gamma \setminus \Sigma$ is the *left endmarker*,
- $\sqcup \in \Gamma \setminus \Sigma$ is the *blank symbol*,
- $s \in Q$ is the *start state*,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,
- $\vdash \in \Gamma \setminus \Sigma$ is the *left endmarker*,
- $\sqcup \in \Gamma \setminus \Sigma$ is the *blank symbol*,
- $s \in Q$ is the *start state*,
- $t \in Q$ is the unique *accepting state*,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,
- $\vdash \in \Gamma \setminus \Sigma$ is the *left endmarker*,
- $\sqcup \in \Gamma \setminus \Sigma$ is the *blank symbol*,
- $s \in Q$ is the *start state*,
- $t \in Q$ is the unique *accepting state*,
- $r \in Q$ $(r \neq t)$ is the unique *rejecting state*,

(Deterministic) Turing Machine

Formal Definition

A TM is $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*, used to encode the input string,
- Γ is a finite *tape alphabet* consisting of symbols written and read from the tape; $\Sigma \subsetneq \Gamma$,
- $\vdash \in \Gamma \setminus \Sigma$ is the *left endmarker*,
- $\sqcup \in \Gamma \setminus \Sigma$ is the *blank symbol*,
- $s \in Q$ is the *start state*,
- $t \in Q$ is the unique *accepting state*,
- $r \in Q$ $(r \neq t)$ is the unique *rejecting state*,
- $\delta : (Q \setminus \{t, r\}) \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition function that never overwrites $\vdash$.

Configuration, and One step

- A configuration of a TM must describe the state, contents of the tape, and position of the head. Thus, $\alpha \in Q \times \{y \sqcup^{\omega} \mid y \in \Gamma^*\} \times \mathbb{N}$.

Configuration, and One step

- A configuration of a TM must describe the state, contents of the tape, and position of the head. Thus, $\alpha \in Q \times \{y \sqcup^\omega \mid y \in \Gamma^*\} \times \mathbb{N}$.
- The starting configuration on input x is $(s, \vdash x \sqcup^\omega, 0)$

Configuration, and One step

- A configuration of a TM must describe the state, contents of the tape, and position of the head. Thus, $\alpha \in Q \times \{y \sqcup^\omega \mid y \in \Gamma^*\} \times \mathbb{N}$.
- The starting configuration on input x is $(s, \vdash x\sqcup^\omega, 0)$
- For a tape $z = y\sqcup^\omega$ $(y \in \Gamma^*)$, $s_b^n(z)$ is the string obtained from z by substituting b for z_n. The next configuration relation is given by

$$\delta(p, z_i) = (q, b, \mathrm{L}) \Rightarrow (p, z, i) \xrightarrow[M]{1} (q, s_b^i(z), i-1),$$
$$\delta(p, z_i) = (q, b, \mathrm{R}) \Rightarrow (p, z, i) \xrightarrow[M]{1} (q, s_b^i(z), i+1).$$

Acceptance, Rejection, and Halting

Let $\xrightarrow[M]{*}$ be the reflexive, transitive closure of $\xrightarrow[M]{1}$.

Acceptance, Rejection, and Halting

Let $\xrightarrow[M]{*}$ be the reflexive, transitive closure of $\xrightarrow[M]{1}$.

- M accepts x if $(s, \vdash x\sqcup^{\omega}, 0) \xrightarrow[M]{*} (t, z, n)$ for some z, n

Acceptance, Rejection, and Halting

Let $\xrightarrow[M]{*}$ be the reflexive, transitive closure of $\xrightarrow[M]{1}$.

- M accepts x if $(s, \vdash x\sqcup^{\omega}, 0) \xrightarrow[M]{*} (t, z, n)$ for some z, n
- M rejects x if $(s, \vdash x\sqcup^{\omega}, 0) \xrightarrow[M]{*} (r, z, n)$ for some z, n

Acceptance, Rejection, and Halting

Let $\xrightarrow[M]{*}$ be the reflexive, transitive closure of $\xrightarrow[M]{1}$.

- M accepts x if $(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (t, z, n)$ for some z, n
- M rejects x if $(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (r, z, n)$ for some z, n
- M does not halt on x if M neither accepts nor rejects x.

Acceptance, Rejection, and Halting

Let $\xrightarrow[M]{*}$ be the reflexive, transitive closure of $\xrightarrow[M]{1}$.

- M accepts x if $(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (t, z, n)$ for some z, n
- M rejects x if $(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (r, z, n)$ for some z, n
- M does not halt on x if M neither accepts nor rejects x.
- M is total if it halts on all inputs x

Language, RE, REC

- Language accepted/recognized by M is $\mathrm{L}(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$.

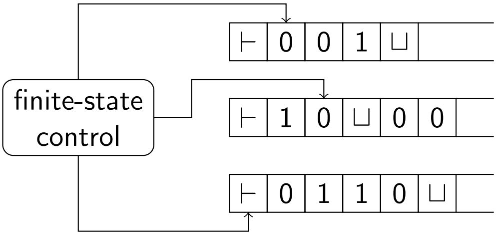
Language, RE, REC

- Language accepted/recognized by M is $\mathrm{L}(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$.
- A language/decision problem L is recursively enumerable (RE) if $L = \mathrm{L}(M)$ for some TM M.

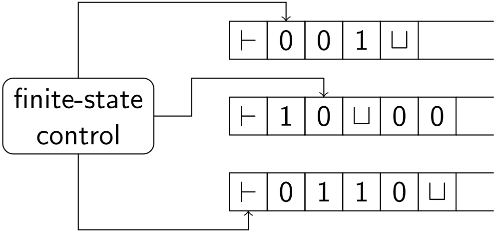
Language, RE, REC

- Language accepted/recognized by M is $\mathrm{L}(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$.
- A language/decision problem L is recursively enumerable (RE) if $L = \mathrm{L}(M)$ for some TM M.
- A language/decision problem L is recursive (REC) if $L = \mathrm{L}(M)$ for some *total* TM M.

Multi-Tape Turing Machine

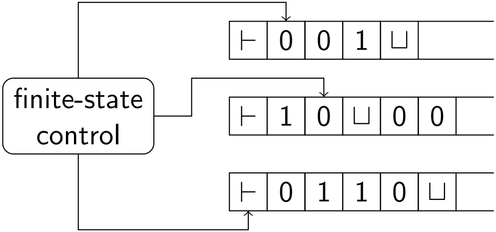


Multi-Tape Turing Machine



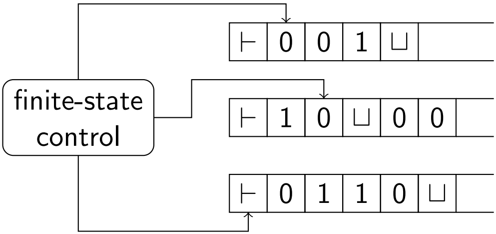
- Input on Tape 1, with left endmarker in cell 0

Multi-Tape Turing Machine



- Input on Tape 1, with left endmarker in cell 0
- Initially all heads scanning cell 0, and tapes 2 to k blank except for the left endmarker

Multi-Tape Turing Machine



- Input on Tape 1, with left endmarker in cell 0
- Initially all heads scanning cell 0, and tapes 2 to k blank except for the left endmarker
- In one step: Read symbols under each of the k-heads, and depending on the current control state, write new symbols on the tapes, move the each tape head (possibly in different directions), and change state.

Expressive Power of multi-tape TM

Expressive Power of multi-tape TM

Theorem

For any k-tape Turing Machine M, there is a single tape TM single(M) *such that* $\mathrm{L}(\mathrm{single}(M)) = \mathrm{L}(M)$.

Expressive Power of multi-tape TM

Theorem

For any k-tape Turing Machine M, there is a single tape TM $\text{single}(M)$ *such that* $\text{L}(\text{single}(M)) = \text{L}(M)$.

Challenges

- How do we store k-tapes in one?

Expressive Power of multi-tape TM

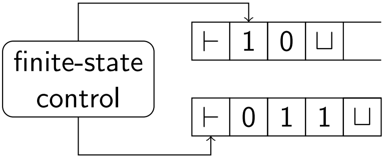
Theorem

For any k-tape Turing Machine M, there is a single tape TM single(M) *such that* L(single(M)) = L(M).

Challenges

- How do we store k-tapes in one?
- How do we simulate the movement of k independent heads?

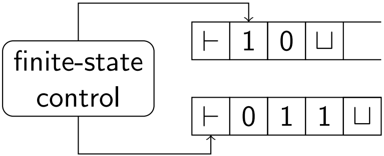
Storing Multiple Tapes



Multi-tape TM M

Store in cell $i+1$ contents of cell i of all tapes.

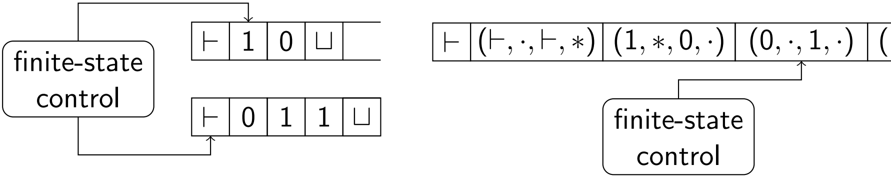
Storing Multiple Tapes



Multi-tape TM M

Store in cell $i+1$ contents of cell i of all tapes. "Mark" head position of tape with $*$.

Storing Multiple Tapes



Multi-tape TM M

1-tape equivalent single(M)

Store in cell $i + 1$ contents of cell i of all tapes. "Mark" head position of tape with $*$.

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

- Read the tape from left to right, storing the contents of the cells being scanned in the state, as we encounter them.

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

- Read the tape from left to right, storing the contents of the cells being scanned in the state, as we encounter them.

Challenge 2: After this scan, 1-tape TM knows the next step of k-tape TM. How do we change the contents and move the heads?

Simulating One Step

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the k symbols that are being read by the k heads, which maybe in different cells?

- Read the tape from left to right, storing the contents of the cells being scanned in the state, as we encounter them.

Challenge 2: After this scan, 1-tape TM knows the next step of k-tape TM. How do we change the contents and move the heads?

- Once again, scan the tape, change all relevant contents, "move" heads (i.e., move $*$s), and change state.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
    - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
    - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
   - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.
   - Scan back from right-to-left moving the heads that need to be moved left.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
   - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.
   - Scan back from right-to-left moving the heads that need to be moved left.

Overall Algorithm

On input w, the 1-tape TM will work as follows.

1. First the machine will rewrite input w to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
   - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.
   - Scan back from right-to-left moving the heads that need to be moved left.

Formal construction in notes.

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic TM: At each step, there are finitely many possibilities. So formally, $N = (Q, \Sigma, \Gamma, \vdash, \sqcup, \Delta, s, t, r)$, where

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic TM: At each step, there are finitely many possibilities. So formally, $N = (Q, \Sigma, \Gamma, \vdash, \sqcup, \Delta, s, t, r)$, where

- $Q, \Sigma, \Gamma, \vdash, \sqcup, s, t, r$ are as before for deterministic machine

Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic TM: At each step, there are finitely many possibilities. So formally, $N = (Q, \Sigma, \Gamma, \vdash, \sqcup, \Delta, s, t, r)$, where

- $Q, \Sigma, \Gamma, \vdash, \sqcup, s, t, r$ are as before for deterministic machine
- $\Delta : (Q \setminus \{t, r\}) \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\xrightarrow[N]{1}$ and multi-step $\xrightarrow[N]{*}$ is defined similarly.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\xrightarrow[N]{1}$ and multi-step $\xrightarrow[N]{*}$ is defined similarly.
- x is accepted by N, if from the starting configuration with x as input, N reaches the accepting state, for some sequence of choices at each step.

Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\xrightarrow[N]{1}$ and multi-step $\xrightarrow[N]{*}$ is defined similarly.
- x is accepted by N, if from the starting configuration with x as input, N reaches the accepting state, for some sequence of choices at each step.
- $\mathrm{L}(N) = \{x \in \Sigma^* \mid N \text{ accepts } x\}$

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine N, there is a (deterministic) TM $\mathrm{det}(N)$ *such that* $\mathrm{L}(\mathrm{det}(N)) = \mathrm{L}(N)$.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine N, there is a (deterministic) TM $\mathrm{det}(N)$ *such that* $\mathrm{L}(\mathrm{det}(N)) = \mathrm{L}(N)$.

Proof Idea

$\mathrm{det}(N)$ will simulate N on the input.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine N, there is a (deterministic) TM $\mathrm{det}(N)$ *such that* $\mathrm{L}(\mathrm{det}(N)) = \mathrm{L}(N)$.

Proof Idea

$\mathrm{det}(N)$ will simulate N on the input.

- Idea 1: $\mathrm{det}(N)$ tries to keep track of all possible "configurations" that N could possibly be after each step.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine N, there is a (deterministic) TM $\text{det}(N)$ *such that* $\text{L}(\text{det}(N)) = \text{L}(N)$.

Proof Idea

$\text{det}(N)$ will simulate N on the input.

- Idea 1: $\text{det}(N)$ tries to keep track of all possible "configurations" that N could possibly be after each step. Works for DFA simulation of NFA

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine N, there is a (deterministic) TM $\mathrm{det}(N)$ *such that* $\mathrm{L}(\mathrm{det}(N)) = \mathrm{L}(N)$.

Proof Idea

$\mathrm{det}(N)$ will simulate N on the input.

- Idea 1: $\mathrm{det}(N)$ tries to keep track of all possible "configurations" that N could possibly be after each step. Works for DFA simulation of NFA but not convenient here.

Expressive Power of Nondeterministic TM

Theorem

For any nondeterministic Turing Machine N, there is a (deterministic) TM $\mathrm{det}(N)$ *such that* $\mathrm{L}(\mathrm{det}(N)) = \mathrm{L}(N)$.

Proof Idea

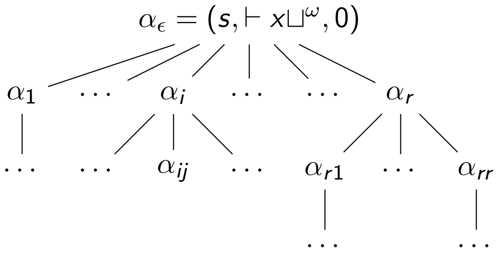
$\mathrm{det}(N)$ will simulate N on the input.

- Idea 1: $\mathrm{det}(N)$ tries to keep track of all possible "configurations" that N could possibly be after each step. Works for DFA simulation of NFA but not convenient here.
- Idea 2: $\mathrm{det}(N)$ will simulate N on each possible sequence of computation steps that N may try in each step.

Nondeterministic Computation

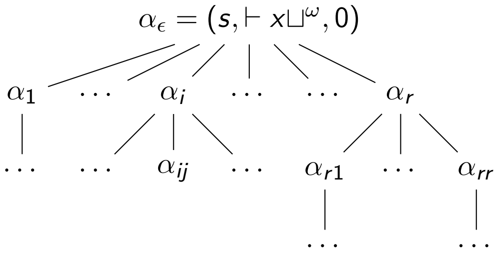
- If $r = \max_{q,X} |\Delta(q, X)|$ then the runs of M can be organized as an r-branching tree.

Nondeterministic Computation



- If $r = \max_{q,X} |\Delta(q, X)|$ then the runs of M can be organized as an r-branching tree.
- $\alpha_{i_1 i_2 \cdots i_n}$ is the configuration of M after n-steps, where choice i_1 is taken in step 1, i_2 in step 2, and so on.

Nondeterministic Computation



- If $r = \max_{q,X} |\Delta(q, X)|$ then the runs of M can be organized as an r-branching tree.
- $\alpha_{i_1 i_2 \cdots i_n}$ is the configuration of M after n-steps, where choice i_1 is taken in step 1, i_2 in step 2, and so on.
- Input x is accepted iff $\exists$ accepting configuration in tree.

Proof Idea

The machine det(N) will search for an accepting configuration in computation tree

Proof Idea

The machine det(N) will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating N on the appropriate sequence of nondeterministic choices

Proof Idea

The machine det(N) will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating N on the appropriate sequence of nondeterministic choices
- det(N) will explore the tree.

Proof Idea

The machine det(N) will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating N on the appropriate sequence of nondeterministic choices
- det(N) will explore the tree.

Observe that det(N) may not terminate if x is not accepted.

Proof Details

det(N) will use 3 tapes to simulate N

Proof Details

det(N) will use 3 tapes to simulate N

Proof Details

det(N) will use 3 tapes to simulate N (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called input tape, will always hold input x

Proof Details

det(N) will use 3 tapes to simulate N (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called input tape, will always hold input x
- Tape 2, called simulation tape, will be used as N's tape when simulating N on a sequence of nondeterministic choices

Proof Details

det(N) will use 3 tapes to simulate N (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called input tape, will always hold input x
- Tape 2, called simulation tape, will be used as N's tape when simulating N on a sequence of nondeterministic choices
- Tape 3, called choice tape, will store the current sequence of nondeterministic choices

Execution of det(N)

1. Initially: Input tape contains x, simulation tape and choice tape are blank
2. Copy contents of input tape onto simulation tape
3. Simulate N using simulation tape as its (only) tape
    1. At the next step of N, if state is q, simulation tape head reads X, and choice tape head reads i, then simulate the ith possibility in $\Delta(q, X)$; if i is not valid, then goto step 4
    2. After changing state, simulation tape contents, and head position on simulation tape, move choice tape's head to the right. If Tape 3 is now scanning $\sqcup$, then goto step 4
    3. If N accepts then accept and halt, else goto step 3(1) to simulate the next step of N.
4. Write the lexicographically next choice sequence on choice tape, erase everything on simulation tape and goto step 2.

Deterministic Simulation

In a nutshell

- det(N) simulates N over and over again, for different sequences, and for different number of steps.

Deterministic Simulation

In a nutshell

- det(N) simulates N over and over again, for different sequences, and for different number of steps.
- If N accepts x then there is a sequence of choices that will lead to acceptance. det(N) will eventually have this sequence on choice tape, and then its simulation N will accept.

Deterministic Simulation

In a nutshell

- det(N) simulates N over and over again, for different sequences, and for different number of steps.
- If N accepts x then there is a sequence of choices that will lead to acceptance. det(N) will eventually have this sequence on choice tape, and then its simulation N will accept.
- If N does not accept x then no sequence of choices leads to acceptance. det(N) will therefore never halt!

Robustness of the Class of TM Languages

Various efforts to capture mechanical computation have the same expressive power.

Robustness of the Class of TM Languages

Various efforts to capture mechanical computation have the same expressive power.

- Non-Turing Machine models: random access machines, λ-calculus, type 0 grammars, first-order reasoning, π-calculus, ...

Robustness of the Class of TM Languages

Various efforts to capture mechanical computation have the same expressive power.

- Non-Turing Machine models: random access machines, λ-calculus, type 0 grammars, first-order reasoning, π-calculus, ...
- Enhanced Turing Machine models: TM with 2-way infinite tape, multi-tape TM, nondeterministic TM, probabilistic Turing Machines, quantum Turing Machines ...

Robustness of the Class of TM Languages

Various efforts to capture mechanical computation have the same expressive power.

- Non-Turing Machine models: random access machines, λ-calculus, type 0 grammars, first-order reasoning, π-calculus, ...
- Enhanced Turing Machine models: TM with 2-way infinite tape, multi-tape TM, nondeterministic TM, probabilistic Turing Machines, quantum Turing Machines ...
- Restricted Turing Machine models: queue machines, 2-stack machines, 2-counter machines, ...

Church-Turing Thesis

"Anything solvable via a mechanical procedure can be solved on a Turing Machine."

Church-Turing Thesis

"Anything solvable via a mechanical procedure can be solved on a Turing Machine."

- Not a mathematical statement that can be proved or disproved!

Church-Turing Thesis

"Anything solvable via a mechanical procedure can be solved on a Turing Machine."

- Not a mathematical statement that can be proved or disproved!
- Strong evidence based on the fact that many attempts to define computation yield the same expressive power

Consequences

- In the course, we will use an informal pseudo-code to argue that a problem/language can be solved on Turing machines

Consequences

- In the course, we will use an informal pseudo-code to argue that a problem/language can be solved on Turing machines
- To show that something can be solved on Turing machines, you can use any programming language of choice, *unless the problem specifically asks you to design a Turing Machine*

Universal Turing Machine

There is a Turing machine U which given the encoding of a Turing machine M and an input x can simulate the execution of M on x and

Universal Turing Machine

There is a Turing machine U which given the encoding of a Turing machine M and an input x can simulate the execution of M on x and (a) Accept if M accepts x, and (b) Reject if M rejects x.

Universal Turing Machine

There is a Turing machine U which given the encoding of a Turing machine M and an input x can simulate the execution of M on x and (a) Accept if M accepts x, and (b) Reject if M rejects x.
U is called the universal Turing machine.

Universal Turing Machine

There is a Turing machine U which given the encoding of a Turing machine M and an input x can simulate the execution of M on x and (a) Accept if M accepts x, and (b) Reject if M rejects x.
U is called the universal Turing machine.
Since U is a fixed machine, its tape alphabet is fixed. However, it needs to be able to simulate TMs with an arbitrary tape alphabet.

Universal Turing Machine

There is a Turing machine U which given the encoding of a Turing machine M and an input x can simulate the execution of M on x and (a) Accept if M accepts x, and (b) Reject if M rejects x.
U is called the universal Turing machine.
Since U is a fixed machine, its tape alphabet is fixed. However, it needs to be able to simulate TMs with an arbitrary tape alphabet. This is achieved by encoding the TMs using a fixed alphabet.

Encoding Turing Machines

Consider an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$.

Encoding Turing Machines

Consider an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$.

- We will encode M using the alphabet $\{0, 1, [,], \bullet, |\}$.

Encoding Turing Machines

Consider an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$.

- We will encode M using the alphabet $\{0, 1, [,], \bullet, |\}$.
- Encode each $a \in \Gamma$ as a binary string enc(a) of length $\log |\Gamma|$; we will assume that $\vdash = 0^{\log |\Gamma|}$ and $\sqcup = 0^{\log |\Gamma| - 1}1$

Encoding Turing Machines

Consider an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$.

- We will encode M using the alphabet $\{0, 1, [,], \bullet, |\}$.
- Encode each $a \in \Gamma$ as a binary string $\text{enc}(a)$ of length $\log |\Gamma|$; we will assume that $\vdash = 0^{\log |\Gamma|}$ and $\sqcup = 0^{\log |\Gamma| - 1}1$
- A string $x = a_1 a_2 \cdots a_n \in \Gamma^*$ will be encoded as $[\text{enc}(a_1) \bullet \text{enc}(a_2) \bullet \cdots \bullet \text{enc}(a_n)]$

Encoding Turing Machines

Consider an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$.

- We will encode M using the alphabet $\{0, 1, [,], \bullet, |\}$.
- Encode each $a \in \Gamma$ as a binary string $\text{enc}(a)$ of length $\log|\Gamma|$; we will assume that $\vdash = 0^{\log|\Gamma|}$ and $\sqcup = 0^{\log|\Gamma|-1}1$
- A string $x = a_1a_2 \cdots a_n \in \Gamma^*$ will be encoded as $[\text{enc}(a_1)\bullet\text{enc}(a_2)\bullet \cdots \bullet\text{enc}(a_n)]$
- Each state $q \in Q$ is encoded as a binary string $\text{enc}(q)$ of length $\log|Q|$; we will assume that $s = 0^{\log|Q|}$, $t = 0^{\log|Q|-1}1$ and $r = 0^{\log|Q|-2}10$

Encoding Turing Machines

Consider an arbitrary Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$.

- We will encode M using the alphabet $\{0, 1, [,], \bullet, |\}$.
- Encode each $a \in \Gamma$ as a binary string enc(a) of length $\log|\Gamma|$; we will assume that $\vdash = 0^{\log|\Gamma|}$ and $\sqcup = 0^{\log|\Gamma|-1}1$
- A string $x = a_1 a_2 \cdots a_n \in \Gamma^*$ will be encoded as $[\text{enc}(a_1)\bullet\text{enc}(a_2)\bullet\cdots\bullet\text{enc}(a_n)]$
- Each state $q \in Q$ is encoded as a binary string enc(q) of length $\log|Q|$; we will assume that $s = 0^{\log|Q|}$, $t = 0^{\log|Q|-1}1$ and $r = 0^{\log|Q|-2}10$
- Directions L and R will be encoded as 0 and 1, respectively.

Turing Machine Codes

Continued

- Each transition $\delta(p, a) = (q, b, d)$ is encoded as $[\text{enc}(p) \bullet \text{enc}(a) | \text{enc}(q) \bullet \text{enc}(b) \bullet \text{enc}(d)]$

Turing Machine Codes

Continued

- Each transition $\delta(p, a) = (q, b, d)$ is encoded as $[\text{enc}(p) \bullet \text{enc}(a) | \text{enc}(q) \bullet \text{enc}(b) \bullet \text{enc}(d)]$
- The code for the machine itself is

$$[\underbrace{\qquad\quad}_{\text{trans 1}}\underbrace{\qquad\quad}_{\text{trans 2}} \cdots\cdots \underbrace{\qquad\quad}_{\text{last trans}}]$$

Turing Machine Codes

Continued

- Each transition $\delta(p, a) = (q, b, d)$ is encoded as $[\text{enc}(p) \bullet \text{enc}(a) | \text{enc}(q) \bullet \text{enc}(b) \bullet \text{enc}(d)]$
- The code for the machine itself is

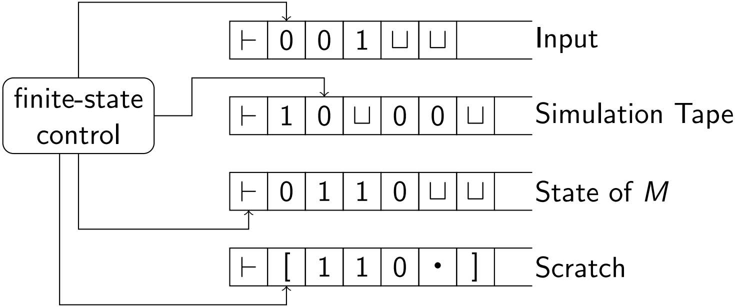
$$[\underbrace{\qquad\qquad}_{\text{trans 1}}\underbrace{\qquad\qquad}_{\text{trans 2}}\cdots\cdots\underbrace{\qquad\qquad}_{\text{last trans}}]$$

- We will denote the encoding of machine M and input x as $\langle M, x \rangle$

Featuring of the Encoding

- The precise choice of the alphabet and encoding is not important; it is merely to illustrate one precise encoding
  - In fact, when we write out TMs on paper using the english alphabet, punctuation marks, and set notation is perfectly good as well, as long as it is consistent.

Universal Turing Machine



Schematic picture of Universal TM

U will store the configuration of M by storing, the state of M on the state tape, and the tape of M on the simulation tape.

UTM algorithm

1. Check to see if the code for M is a valid TM code; if not reject the input. For example, for our code, it involves checking to see if the string as exactly one “|” between [and], etc.

UTM algorithm

1. Check to see if the code for M is a valid TM code; if not reject the input. For example, for our code, it involves checking to see if the string as exactly one "|" between [and], etc.
2. If code is ok, then copy x onto tape 2

UTM algorithm

1. Check to see if the code for M is a valid TM code; if not reject the input. For example, for our code, it involves checking to see if the string as exactly one "|" between [and], etc.
2. If code is ok, then copy x onto tape 2
3. Write $0 \cdots 0$, the start state of M, on the third tape, and scan the "first cell" of tape 2.

UTM algorithm

1. Check to see if the code for M is a valid TM code; if not reject the input. For example, for our code, it involves checking to see if the string as exactly one "|" between [and], etc.
2. If code is ok, then copy x onto tape 2
3. Write $0 \cdots 0$, the start state of M, on the third tape, and scan the "first cell" of tape 2.
4. To simulate a move of M, search for a transition [enc(p)•enc(a)|enc(q)•enc(b)•enc(d)] on tape 1, where enc(p) is on tape 3 (current state) and enc(a) is read from tape 2 from the current "cell", i.e., between two successive • symbols from the current head position. Then, write enc(q) on tape 3 (after erasing its current contents), write enc(b) on tape 2 instead of enc(a), and finally move the head on tape 2 to the appropriate "cell".

UTM algorithm

1. Check to see if the code for M is a valid TM code; if not reject the input. For example, for our code, it involves checking to see if the string as exactly one "|" between [and], etc.
2. If code is ok, then copy x onto tape 2
3. Write $0 \cdots 0$, the start state of M, on the third tape, and scan the "first cell" of tape 2.
4. To simulate a move of M, search for a transition [enc(p)•enc(a)|enc(q)•enc(b)•enc(d)] on tape 1, where enc(p) is on tape 3 (current state) and enc(a) is read from tape 2 from the current "cell", i.e., between two successive • symbols from the current head position. Then, write enc(q) on tape 3 (after erasing its current contents), write enc(b) on tape 2 instead of enc(a), and finally move the head on tape 2 to the appropriate "cell".
5. If state on tape 3 is $0 \cdots 01$ then accept; if state is $0 \cdots 010$ then reject.