CS 611: Theory of Computation

Hongmin Li

Department of Computer Science
California State University, East Bay

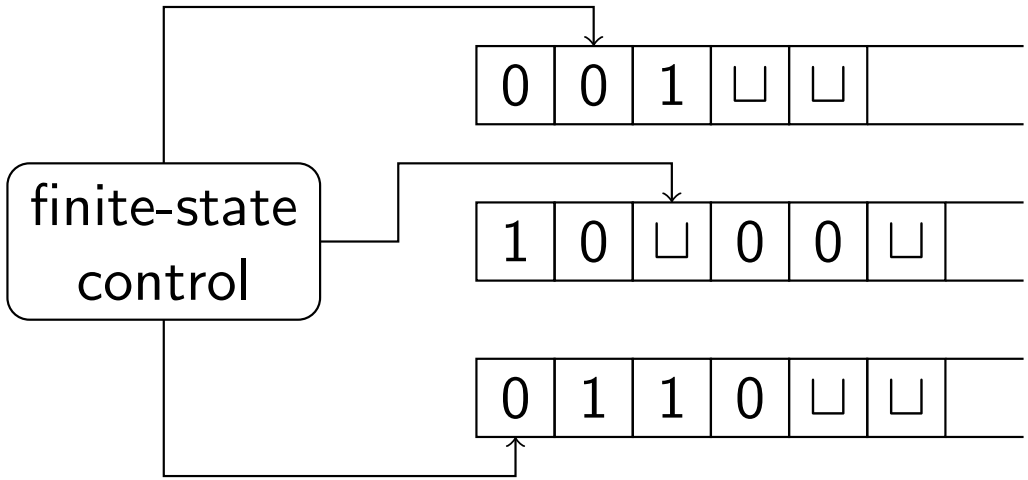
Turing Machines

Formal Definition

A Turing machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ where

- Q is a finite set of control states
- Σ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$
- $q_0 \in Q$ is the initial state
- $q_{\text{acc}} \in Q$ is the accept state
- $q_{\text{rej}} \in Q$ is the reject state, where $q_{\text{rej}} \neq q_{\text{acc}}$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function. Given the current state and symbol being read, the transition function describes the next state, symbol to be written and direction (left or right) in which to move the tape head.

Multi-Tape Turing Machine



- Input on Tape 1
- Initially all heads scanning cell 1, and tapes 2 to k blank
- 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.

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, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$, 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, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$, where

- $Q, \Sigma, \Gamma, q_0, q_{\text{acc}}, q_{\text{rej}}$ are as before for 1-tape 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, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$, where

- $Q, \Sigma, \Gamma, q_0, q_{\text{acc}}, q_{\text{rej}}$ are as before for 1-tape machine
- $\delta : (Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\})$

Turing Machine

Accept and Reject

On input w a TM M may halt (enter q_{acc} or q_{rej}) or M may run forever ("loop"). So M has 3 possible outcomes for each input w:

- Accept w (enter q_{acc})
- Reject w by halting (enter q_{rej})
- Reject w by looping (running forever)

Turing Machine

TM Recognizers and Deciders

Definition

For a Turing machine M, define $L(M) = \{w \mid M \text{ accepts } w\}$. M is said to accept or recognize a language L if $L = L(M)$.

Definition

A Turing machine M is said to decide a language L if $L = L(M)$ and M halts on every input.

Turing Machine

TM Recognizers and Deciders

Definition

L is Turing-recognizable if $L = L(M)$ for some TM M.

Definition

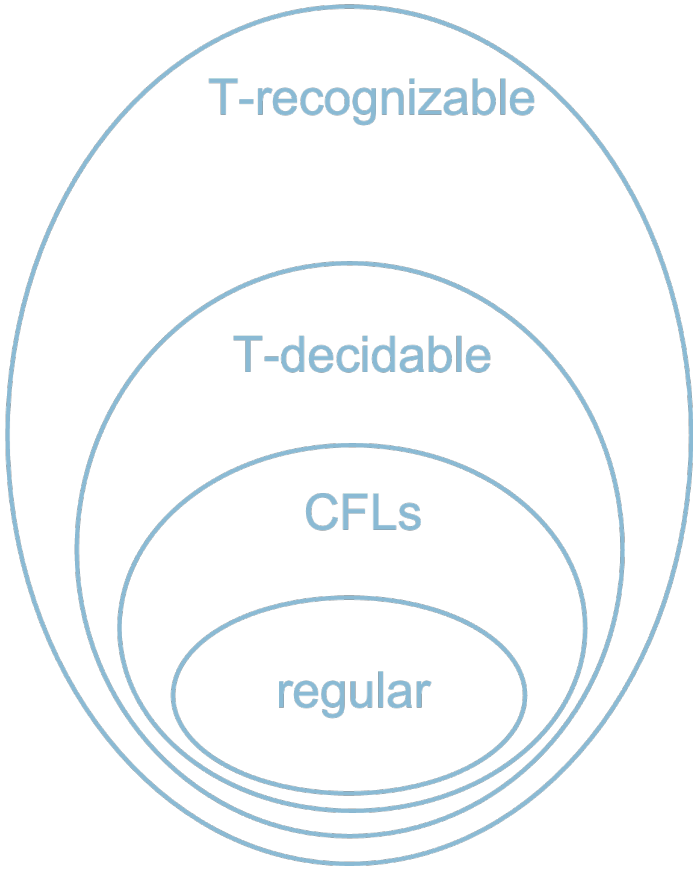
TM M is a decider if M halts on all inputs.

Say that M decides L if $L = L(M)$ and M is decider.

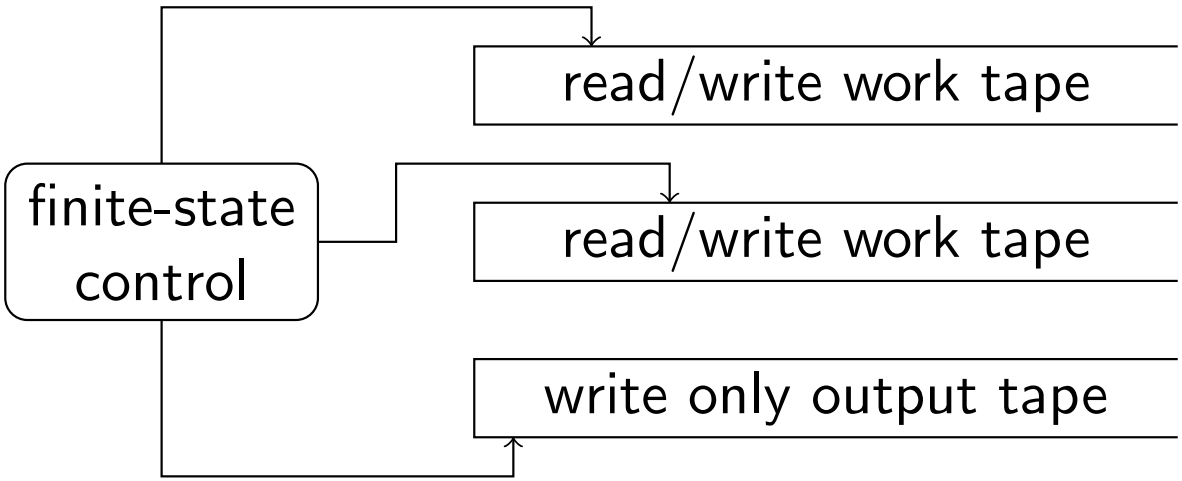
Definition

L is Turing-decidable if TM M is a decider if M halts on all inputs.

Big Picture

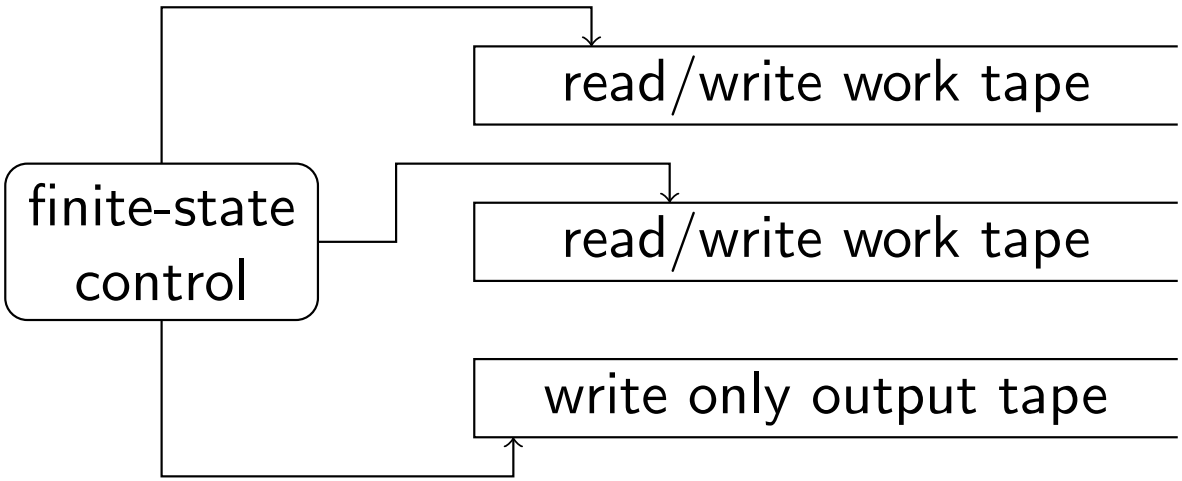


Enumerators



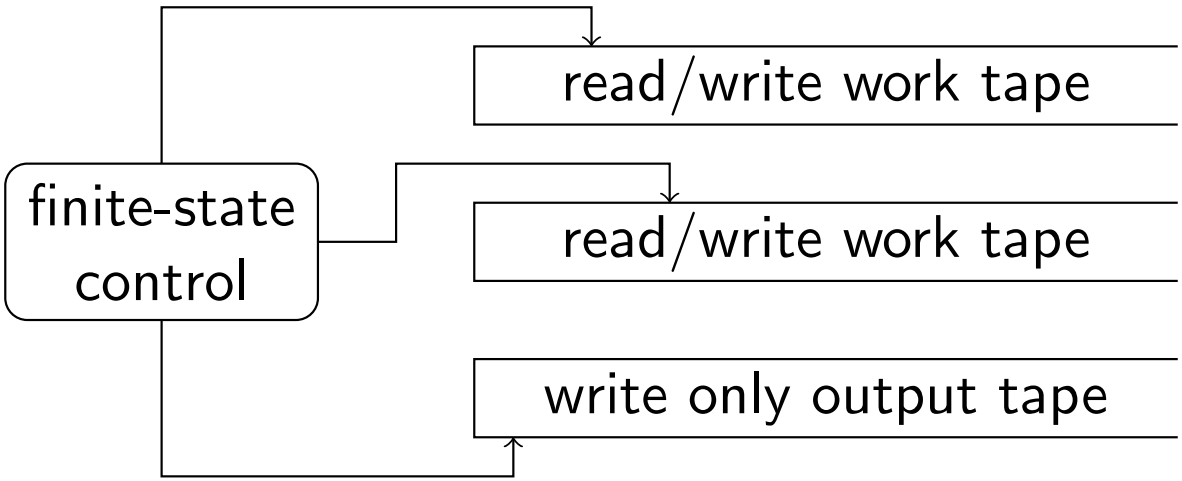
- An enumerator is multi-tape Turing Machine, with a special output tape which is write-only
  - Write-only means (a) symbol on output tape does not affect transitions, and (b) tape head only moves right.

Enumerators



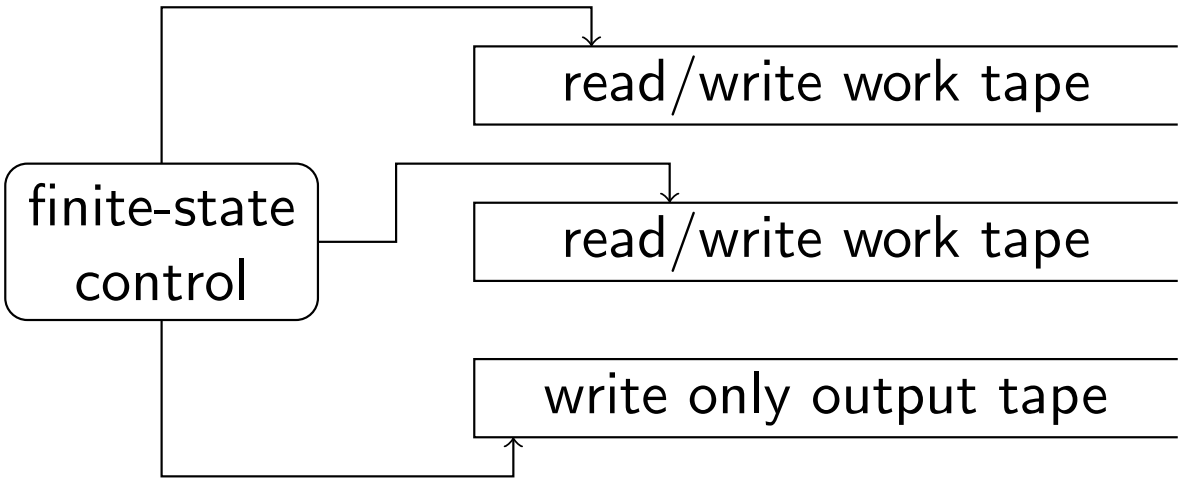
- An enumerator is multi-tape Turing Machine, with a special output tape which is write-only
  - Write-only means (a) symbol on output tape does not affect transitions, and (b) tape head only moves right.
- Intially all tapes blank (no input).

Enumerators



- An enumerator is multi-tape Turing Machine, with a special output tape which is write-only
  - Write-only means (a) symbol on output tape does not affect transitions, and (b) tape head only moves right.
- Intially all tapes blank (no input). During computation the machine adds symbols to the output tape.

Enumerators



- An enumerator is multi-tape Turing Machine, with a special output tape which is write-only
  - Write-only means (a) symbol on output tape does not affect transitions, and (b) tape head only moves right.
- Intially all tapes blank (no input). During computation the machine adds symbols to the output tape. Output considered to be a list of words (separated by special symbol #)

Recursively Enumerable Languages

Definition

An enumerator M is said to enumerate a string w if and only if at some point M writes a word w on the output tape.

Recursively Enumerable Languages

Definition

An enumerator M is said to enumerate a string w if and only if at some point M writes a word w on the output tape.
$E(M) = \{w \mid M \text{ enumerates } w\}$

Recursively Enumerable Languages

Definition

An enumerator M is said to enumerate a string w if and only if at some point M writes a word w on the output tape.
$E(M) = \{w \mid M \text{ enumerates } w\}$

Note

M need not enumerate strings in order. It is also possible that M lists some strings many times!

Recursively Enumerable Languages

Definition

An enumerator M is said to enumerate a string w if and only if at some point M writes a word w on the output tape.
$E(M) = \{w \mid M \text{ enumerates } w\}$

Note

M need not enumerate strings in order. It is also possible that M lists some strings many times!

Definition

L is recursively enumerable (r.e.) iff there is an enumerator M such that $L = E(M)$.

Recursively Enumerable Languages and TMs

Theorem

L is recursively enumerable if and only if L is Turing-recognizable.

Recursively Enumerable Languages and TMs

Theorem

L is recursively enumerable if and only if L is Turing-recognizable.

Note

Hence, when we say a language L is recursively enumerable (r.e.) then

- there is a TM that accepts L, and
- there is an enumerator that enumerates L.

Recognizers From Enumerators

Proof.

Suppose L is enumerated by N. Need to construct M such that $L(M) = E(N)$.

Recognizers From Enumerators

Proof.

Suppose L is enumerated by N. Need to construct M such that $L(M) = E(N)$. M is the following TM

On input w
    Run N. Every time N writes a word 'x'
    compare x with w.
    If $x = w$ then accept and halt
    else continue simulating N

Recognizers From Enumerators

Proof.

Suppose L is enumerated by N. Need to construct M such that $L(M) = E(N)$. M is the following TM

```
On input w
    Run N.  Every time N writes a word 'x'
    compare x with w.
    If x = w then accept and halt
    else continue simulating N
```

Clearly, if $w \in L$, M accepts w, and if $w \notin L$ then M never halts. $\cdots\rightarrow$

Enumerators From Recognizers

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$.

Enumerators From Recognizers

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

for $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$ do
    simulate M on w
    if M accepts w then write the word 'w'
        on output tape

Enumerators From Recognizers?

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

```
for w = ε, 0, 1, 00, 01, 10, 11, 000, ... do
    simulate M on w
    if M accepts w then write the word 'w'
        on output tape
```

Does N enumerate L?

Enumerators From Recognizers?

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

```
for w = ε, 0, 1, 00, 01, 10, 11, 000, ... do
    simulate M on w
    if M accepts w then write the word 'w'
        on output tape
```

Does N enumerate L? No!! M may not halt on a string $w \notin L$, in which case N will not output any more strings!

Enumerators From Recognizers?

Parallel simulation

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

```
for w = ϵ, 0, 1, 00, 01, 10, 11, 000, ... do
    simulate M on w
    if M accepts w then write the word 'w'
        on output tape
```

Does N enumerate L? No!! M may not halt on a string $w \notin L$, in which case N will not output any more strings!
Must simulate M on all inputs in parallel.

Enumerators From Recognizers?

Parallel simulation?

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

```
for w = ϵ, 0, 1, 00, 01, 10, 11, 000, ... do
    simulate M on w
    if M accepts w then write the word 'w'
        on output tape
```

Does N enumerate L? No!! M may not halt on a string $w \notin L$, in which case N will not output any more strings!
Must simulate M on all inputs in parallel. But infinitely many parallel executions.

Enumerators From Recognizers?

Parallel simulation?

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

```
for w = ϵ, 0, 1, 00, 01, 10, 11, 000, ... do
    simulate M on w
    if M accepts w then write the word 'w'
        on output tape
```

Does N enumerate L? No!! M may not halt on a string $w \notin L$, in which case N will not output any more strings!
Must simulate M on all inputs in parallel. But infinitely many parallel executions. Will never reach step two in any execution!

Enumerators From Recognizers

Dovetailing

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$.

Enumerators From Recognizers

Dovetailing

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

for $i = 1, 2, 3 \ldots$ do
    let $w_1, w_2, \ldots w_i$ be the first i strings (in lexicographic order)
    simulate M on w_1 for i steps, then on w_2 for i steps and ...simulate M on w_i for i steps
    if M accepts w_j within i steps then write w_j (with separator) on output tape

Enumerators From Recognizers

Dovetailing

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

for $i = 1, 2, 3 \ldots$ do
    let $w_1, w_2, \ldots w_i$ be the first i strings (in lexicographic order)
    simulate M on w_1 for i steps, then on w_2 for i steps and ...simulate M on w_i for i steps
    if M accepts w_j within i steps then write w_j (with separator) on output tape

Observe that $w \in L(M)$ iff N will enumerates w.

Enumerators From Recognizers

Dovetailing

Proof (contd).

Let M be such that $L = L(M)$. Need to construct N such that $E(N) = L(M)$. N is the following enumerator

for $i = 1, 2, 3 \ldots$ do
    let $w_1, w_2, \ldots w_i$ be the first i strings (in lexicographic order)
    simulate M on w_1 for i steps, then on w_2 for i steps and ...simulate M on w_i for i steps
    if M accepts w_j within i steps then write w_j (with separator) on output tape

Observe that $w \in L(M)$ iff N will enumerates w. N will enumerate strings many times!

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

"Intuitive notion of Algorithms = 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!
- Strong evidence based on the fact that many attempts to define computation yield the same expressive power

"Intuitive notion of Algorithms = Turing Machine."

- Connection between the informal notion of algorithm and the precise definition
- Definition of algorithm necessary to resolve Hilbert's tenth problem

Hibert's 10th problem

In 1900 David Hilbert posed 23 problems:

- Give an algorithm for solving Diophantine equations.

Diophantine equations:

Equations of polynomials where solutions must be integers.
Example:

$$3x^2 - 2xy - y^2z = 7 \tag{1}$$

Solution: $x = 1, y = 2, z = -2$

Let
$D = \{p | \text{polynomial} p(x_1, x_2, \ldots, x_k) = 0$ has a solution in integers$\}$
Hilbert's 10th problem: Give an algorithm to decide D.
Matiyasevich proved in 1970: D is not decidable.
Note: D is T-recognizable.

Notation for encodings and TMs

Notation for encodings and TMs

- If O is some object (e.g., polynomial, automaton, graph, etc.), we write $\langle O \rangle$ to be an encoding of that object into a string.
- If $O_1, O_2, \ldots, O_k$ is a list of objects then we write $\langle O_1, O_2, \ldots, O\_k \rangle$ to be an encoding of them together into a single string.

Notation for writing Turing machines:
We will use high-level English descriptions of algorithms when we describe TMs, knowing that we could (in principle) convert those descriptions into states, transition function, etc. Our notation for writing a TM M is

$$M = \text{“On input } w \tag{2}$$

$$\text{[English description of the algorithm]”} \tag{3}$$

TM - example revisited

TM M recognizing $B = \{a^k b^k c^k | k \geq 0\}$

$M =$ "On input w (4)

1. Check if $w \in a^* b^* c^*$, reject if not (5)
2. Count the number of a, b, c in w (6)
3. Accept if all counts are equal; reject if not. (7)

High-level description is ok. You do not need to manage tapes, states, etc ...