Limits of Turing Machines (Part 2)

Dr. Chuck Rocca
roccac@wcsu.edu
http://sites.wcsu.edu/roccac

WESTERN CONNECTICUT STATE UNIVERSITY
MACRICOSTAS SCHOOL OF ARTS & SCIENCES

Table of Contents

Table of Contents

Decidable vs. Recognizable

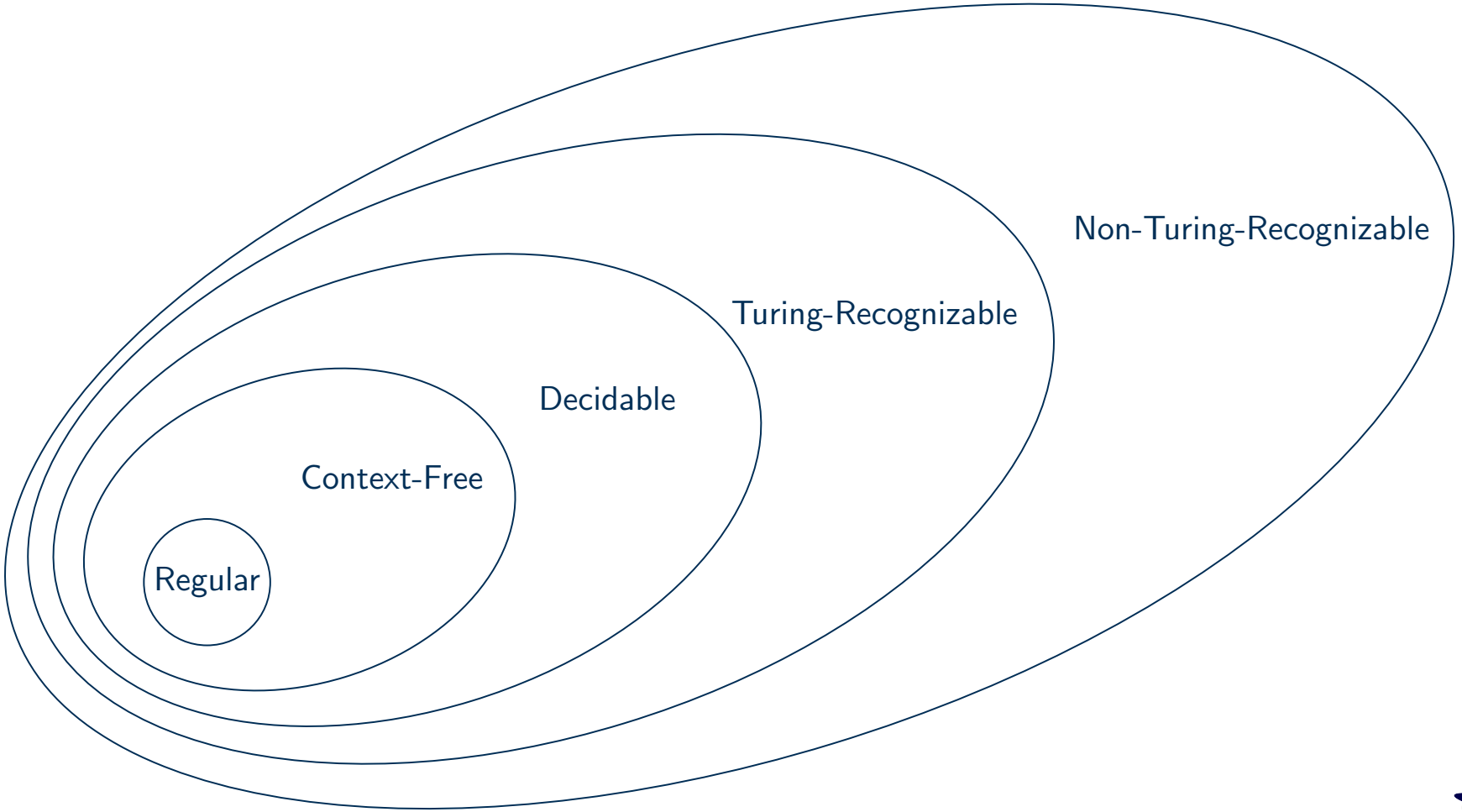
Definition

A Language is *Turning-decidable* or simply *decidable* if some Turing machine decides it; the machine always reaches an accept or reject state. Given any word there is a TM that can tell if the word is or is not in the language.

Definition

A Language is *Turning-recognizable* if some Turing machine recognizes it; in this case the machine reaches an accept state, reject state, or it may loop (fail to accept). There is a TM that accepts words in the language, but may fail to reach a verdict if a word is not in the language.

Hierarchy of Languages



Deterministic Finite Automaton

Theorem

The set

$$A_{DFA} = \{\langle B, w\rangle \,|\, B \text{ is a DFA that accepts string } w\}$$

is a decidable language.

M="On input $\langle B, w\rangle$, where B is a DFA and w is a string:

0. Check the format of the input.
1. Simulate B on input w.
2. If the simulation ends in an accept state, *accept*; otherwise, *reject*."

Empty Languages

Theorem

The set

$$E_{DFA} = \{\langle A \rangle \,|\, A \text{ is a DFA and } L(A) = \emptyset\}$$

is a decidable language.

T="On input $\langle A \rangle$, the string encoding of DFA A:

1. Select the start state in A and mark it.
2. Repeat the following until no new states are marked:
3. For each state in A, mark it if there is a transition from a marked state.
4. Scan the accept states of A, if any are marked, *reject*; otherwise *accept*."

Equal Languages

Theorem

The set

$$EQ_{DFA} = \{\langle A, B\rangle \,|A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

is a decidable language.

- Note, DFAs are closed under *unions*, *intersections*, and *compliments*.
- Construct the *symmetric difference* of A and B, $C \equiv A \text{ XOR } B$ or

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

- Check if C is empty using T from the previous theorem.

Equal Languages

Theorem

The set

$$EQ_{DFA} = \{\langle A, B\rangle \,|A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

is a decidable language.

- Note, DFAs are closed under *unions*, *intersections*, and *compliments*.
- Construct the *symmetric difference* of A and B, $C \equiv A \text{ XOR } B$ or

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

- Check if C is empty using T from the previous theorem. (Reduction)

Nondeterministic Finite Automaton

Theorem

The set

$$A_{NFA} = \{\langle B, w\rangle \,|\, B \text{ is an NFA that accepts string } w\}$$

is a decidable language.

N="On input $\langle B, w\rangle$, where B is a NFA and w is a string:

1. Convert B to a DFA C. (Reduction)
2. Run M from the previous theorem on input $\langle C, w\rangle$.
3. If the simulation ends in an accept state, *accept*; otherwise, *reject*."

Regular Expressions

Theorem

The set

$$A_{REX} = \{\langle B, w\rangle \,|\, B \text{ is a regex that accepts string } w\}$$

is a decidable language.

P="On input $\langle B, w\rangle$, where B is a RegEx and w is a string:

1. Convert B to a NFA C. (Reduction)
2. Run N from the previous theorem on input $\langle C, w\rangle$.
3. If the simulation ends in an accept state, *accept*; otherwise, *reject*."

CFLs

Theorem

The set

$$A_{CFG} = \{\langle G, w\rangle \,|\, G \text{ is a CFG that generates the string } w\}$$

is a decidable language.

$S =$"On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G to Chomsky Normal Form. (Reduction)
2. List all derivations with $2n - 1$ steps, $n = |w|$; except for n=0, then list derivations with one step.
3. If w is generated, *accept*; otherwise *reject*."

CFLs

Theorem

The set

$$E_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \emptyset\}$$

is a decidable language.

R="On input $\langle G \rangle$, the string encoding of CFG G:

1. Mark the terminals in G.
2. Repeat the following until no new variables get marked:
3. Mark any variable A where $A \to U_1 U_2 \cdots U_k$ is in G and the U_i are all marked.
4. If the start variable of G is not marked, *accept*; otherwise *reject*."

CFLs

Theorem

The set

$$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

is not a decidable language. (Proof held until after Chapter 5.)

Table of Contents

A_{TM} is Undecidable

Theorem

Given the set

$$A_{TM} = \{\langle M, w\rangle \,|\, M \text{ is a Turing Machine and } M \text{ accepts } w\},$$

A_{TM} is undecidable.

$U =$ " On input $\langle M, w\rangle$, where M is a TM and w is a string:

1. Simulate M on w.
2. If M ever enters its accept state, *accept*; If M ever enters its reject state, *reject*."

This is a *universal Turing machine* and shows that A_{TM} is recognizable.

An Undecidable Language

- $A_{TM} = \{\langle M, w\rangle \,|\, M \text{ is a Turing Machine and } M \text{ accepts } w\}$

An Undecidable Language

- $A_{TM} = \{\langle M, w\rangle \,|\, M \text{ is a Turing Machine and } M \text{ accepts } w\}$
- Suppose there's a *decider* H for A_{TM}

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

An Undecidable Language

- $A_{TM} = \{\langle M, w\rangle \,|\, M \text{ is a Turing Machine and } M \text{ accepts } w\}$
- Suppose there's a *decider* H for A_{TM}
- Define $D(\langle M\rangle) = \neg H(\langle M, \langle M\rangle\rangle)$

$$D(\langle M\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

An Undecidable Language

- $A_{TM} = \{\langle M, w\rangle \,|\, M \text{ is a Turing Machine and } M \text{ accepts } w\}$
- Suppose there's a *decider* H for A_{TM}
- Define $D(\langle M\rangle) = \neg H(\langle M, \langle M\rangle\rangle)$
- D can't decide $\langle D\rangle$

$$D(\langle D\rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D\rangle \end{cases}$$

An Undecidable Language

- $A_{TM} = \{\langle M, w\rangle \,|\, M \text{ is a Turing Machine and } M \text{ accepts } w\}$
- Suppose there's a *decider* H for A_{TM}
- Define $D(\langle M\rangle) = \neg H(\langle M, \langle M\rangle\rangle)$
- D can't decide $\langle D\rangle$

$$D(\langle D\rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D\rangle \end{cases}$$

- $\therefore$ Neither D nor H can exist and so A_{TM} is undecidable

Machine D

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$
M_1	*accept*	*reject*	*accept*	*accept*	$\cdots$	*accept*
M_2	*reject*	*accept*	*accept*	*reject*	$\cdots$	*reject*
M_3	*accept*	*accept*	*reject*	*reject*	$\cdots$	*reject*
M_4	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\cdots$
D	*reject*	*reject*	*accept*	*reject*	$\cdots$	?

A_{TM} is Undecidable

Theorem

Given the set

$$A_{TM} = \{\langle M, w\rangle \,|\, M \text{ is a Turing Machine and } M \text{ accepts } w\},$$

A_{TM} *is undecidable.*

Non-Turing Recognizable Languages

Theorem

There exist languages that are not Turing-Recognizable.

- $\Sigma^* = \{s_1, s_2, s_3, s_4, \ldots\}$
- $\mathscr{B} = \{\text{infinite binary sequences}\}$
- $\mathscr{L} = \{\text{all languages}\} = \mathscr{P}(\Sigma^*)$
- $f : \mathscr{L} \longrightarrow \mathscr{B}$
- $\forall A \in \mathscr{L} : f(A) = b_1 b_2 b_3 b_4 \cdots \in \mathscr{B}$

$$b_i = \begin{cases} 0 & s_i \notin A \\ 1 & s_i \in A \end{cases}$$

- $\chi_A = f(A)$ is called the *characteristic sequence* of A

Decidability vs. Recognizably

Definition

A language, $\overline{A}$, is *co-Turing-recognizable* if it is the complement of a Turing-recognizable language A.

Decidability vs. Recognizably

Definition

A language, $\overline{A}$, is *co-Turing-recognizable* if it is the complement of a Turing-recognizable language A.

Theorem

A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable. (i.e. A and $\overline{A}$ are both recognizable)

Decidability vs. Recognizably

Definition

A language, $\overline{A}$, is *co-Turing-recognizable* if it is the complement of a Turing-recognizable language A.

Theorem

A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable. (i.e. A and $\overline{A}$ are both recognizable)

Theorem

A language is undecidable if and only if it is not Turing-recognizable or not co-Turing-recognizable.

Decidability vs. Recognizably

Definition

A language, $\overline{A}$, is *co-Turing-recognizable* if it is the complement of a Turing-recognizable language A.

Theorem

A language is undecidable if and only if it is not Turing-recognizable or not co-Turing-recognizable.

Corollary

The language $\overline{A_{TM}}$ is not Turing-recognizable.

Table of Contents

Reducible Languages

Definition (Reducible)

A *reduction* of a problem is a process of converting one problem to another in such a way that a solution to the second problem provides a solution to the first.

Reducible Languages

Definition (Reducible)

A *reduction* of a problem is a process of converting one problem to another in such a way that a solution to the second problem provides a solution to the first.

- Primality testing can be reduced to factoring. (But not necessarily the other way.)

Reducible Languages

Definition (Reducible)

A *reduction* of a problem is a process of converting one problem to another in such a way that a solution to the second problem provides a solution to the first.

- Primality testing can be reduced to factoring. (But not necessarily the other way.)
- Decidability of A_{REX} reduces to the decidability of A_{NFA}.

Reducible Languages

Definition (Reducible)

A *reduction* of a problem is a process of converting one problem to another in such a way that a solution to the second problem provides a solution to the first.

- Primality testing can be reduced to factoring. (But not necessarily the other way.)
- Decidability of A_{REX} reduces to the decidability of A_{NFA}.
- Decidability of A_{NFA} reduces to the decidability of A_{DFA}.

Reducible Languages

Definition (Reducible)

A *reduction* of a problem is a process of converting one problem to another in such a way that a solution to the second problem provides a solution to the first.

- Primality testing can be reduced to factoring. (But not necessarily the other way.)
- Decidability of A_{REX} reduces to the decidability of A_{NFA}.
- Decidability of A_{NFA} reduces to the decidability of A_{DFA}.
- Decidability of EQ_{DFA} reduces to the decidability of E_{DFA}.

Mapping Reducible

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Mapping Reducible

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Example

- Arithmetic operations on $\langle x, y \rangle$ such as $x + y$.
- Converting a *TM* $\langle M \rangle$ into a *TM* $\langle M' \rangle$ which marks the start of a string w and never tries to move left from the left-hand end of the tape.

Mapping Reducible

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Definition

Language A is *mapping reducible* to a language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B.$$

The function f is called the *reduction* of A to B.

Recasting a Previous Examples

Example

Let M_1 be a DFA that rejects everything. Then

$$\langle M \rangle \mapsto \langle M, M_1 \rangle$$

is a mapping reduction from E_{DFA} to EQ_{DFA}. That is $L(M)$ is the empty language if and only if $L(M) = L(M_1)$, and thus $E_{DFA} \leq_m EQ_{DFA}$.

Decidable or Not

Theorem

Given $A \leq_m B$ (A is a reducible to B), if B is decidable, then A is decidable.

Decidable or Not

Theorem

Given $A \leq_m B$ (A is a reducible to B), if B is decidable, then A is decidable.

Theorem (Contrapositive)

Given $A \leq_m B$ (A is a reducible to B), if A is undecidable, then B is undecidable.

Recognizable or Not

Theorem

Given $A \leq_m B$ (A is a reducible to B), if B is Turing-recognizable, then A is Turing-recognizable.

Recognizable or Not

Theorem

Given $A \leq_m B$ (A is a reducible to B), if B is Turing-recognizable, then A is Turing-recognizable.

Theorem (Contrapositive)

Given $A \leq_m B$ (A is a reducible to B), if A is not Turing-recognizable, then B is not Turing-recognizable.

Rice's Theorem

Theorem (Rice's Theorem)

Let P be any non-trivial property of the language of some Turing machine, i.e.

$$P = \{\langle M \rangle \,|\, M \text{ is a TM and } L(M) \text{ has property } P\},$$

such that

1. *some but not all TMs are elements of P, and*
2. *if $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$,*

then P is undecidable.

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P

$\square$

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:

$\square$

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_w:
     M_w="On input x:

□

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_w:
     M_w="On input x:
     1. Simulate M on w. If it halts or rejects, then *reject*.

□

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_w:
     M_w="On input x:
     1. Simulate M on w. If it halts or rejects, then *reject*.
     2. Simulate T on x. If it accepts, then *accept*."

□

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_w:
     M_w="On input x:
     1. Simulate M on w. If it halts or rejects, then *reject*.
     2. Simulate T on x. If it accepts, then *accept*."
  2. Use *TM* R_P to determine if $\langle M_w \rangle \in P$. If yes, *accept*. If no, *reject*."

□

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_w:
     M_w="On input x:
     1. Simulate M on w. If it halts or rejects, then *reject*.
     2. Simulate T on x. If it accepts, then *accept*."
  2. Use *TM* R_P to determine if $\langle M_w \rangle \in P$. If yes, *accept*. If no, *reject*."
- $L(M_w) = \emptyset$ implies $\langle M, w \rangle \notin A_{TM}$

$\square$

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_w:
     M_w="On input x:
     1. Simulate M on w. If it halts or rejects, then *reject*.
     2. Simulate T on x. If it accepts, then *accept*."
  2. Use *TM* R_P to determine if $\langle M_w \rangle \in P$. If yes, *accept*. If no, *reject*."
- $L(M_w) = \emptyset$ implies $\langle M, w \rangle \notin A_{TM}$
- $L(M_w) = L(T) \in P$ implies $\langle M, w \rangle \in A_{TM}$

$\square$

Rice's Theorem Proof (by contradiction)

Proof.

- Assume $\langle T \rangle \in P$ and R_P is a decider for P
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_w:
     M_w="On input x:
     1. Simulate M on w. If it halts or rejects, then *reject*.
     2. Simulate T on x. If it accepts, then *accept*."
  2. Use *TM* R_P to determine if $\langle M_w \rangle \in P$. If yes, *accept*. If no, *reject*."
- $L(M_w) = \emptyset$ implies $\langle M, w \rangle \notin A_{TM}$
- $L(M_w) = L(T) \in P$ implies $\langle M, w \rangle \in A_{TM}$
- S is a decider for A_{TM}, this is a contradiction $\square$

Table of Contents

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M \text{ halts on input } w\}$

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M \text{ halts on input } w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \,|\, TM\ M$ halts on input $w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.
- S="On input $\langle M, w\rangle$:

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M$ halts on input $w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.
- S="On input $\langle M, w\rangle$:
  1. Run $TM\ R$ on input $\langle M, w\rangle$.

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M$ halts on input $w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.
- S="On input $\langle M, w\rangle$:
  1. Run $TM\ R$ on input $\langle M, w\rangle$.
  2. if R rejects, *reject*.

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M \text{ halts on input } w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.
- S="On input $\langle M, w\rangle$:
  1. Run $TM\ R$ on input $\langle M, w\rangle$.
  2. if R rejects, *reject*.
  3. If R accepts, simulate M on w until it halts.

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M \text{ halts on input } w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.
- S="On input $\langle M, w\rangle$:
  1. Run $TM\ R$ on input $\langle M, w\rangle$.
  2. if R rejects, *reject*.
  3. If R accepts, simulate M on w until it halts.
  4. If M accepts, *accept*; if M rejects, *reject*."

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M \text{ halts on input } w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.
- S="On input $\langle M, w\rangle$:
  1. Run $TM\ R$ on input $\langle M, w\rangle$.
  2. if R rejects, *reject*.
  3. If R accepts, simulate M on w until it halts.
  4. If M accepts, *accept*; if M rejects, *reject*."
- S is a decider for A_{TM}, this is a contradiction.

The Halting Problem, $HALT_{TM}$ is Undecidable

- $HALT_{TM} = \{\langle M, w\rangle \mid TM\ M$ halts on input $w\}$
- Assume $TM\ R$ decides $HALT_{TM}$.
- S="On input $\langle M, w\rangle$:
  1. Run $TM\ R$ on input $\langle M, w\rangle$.
  2. if R rejects, *reject*.
  3. If R accepts, simulate M on w until it halts.
  4. If M accepts, *accept*; if M rejects, *reject*."
- S is a decider for A_{TM}, this is a contradiction.
- $\therefore$ $HALT_{TM}$ is undecidable.

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \mid L(M) \text{ is a regular language}\}$

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M) \text{ is a regular language}\}$
- Assume *TM R* decides $REGULAR_{TM}$.

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM* R decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM* R decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_1:
     M_1="On input x:

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM* R decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_1:
     M_1="On input x:
     1. If $x = 0^n 1^n$, *accept*. (Recall $0^n 1^n$ is non-regular.)

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM* R decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_1:
     M_1="On input x:
     1. If $x = 0^n 1^n$, *accept*. (Recall $0^n 1^n$ is non-regular.)
     2. If $x \neq 0^n 1^n$, run M on w and *accept* if M accepts w."

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM R* decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_1:
     M_1="On input x:
     1. If $x = 0^n 1^n$, *accept*. (Recall $0^n 1^n$ is non-regular.)
     2. If $x \neq 0^n 1^n$, run M on w and *accept* if M accepts w."
  2. Use *TM R* to determine if $\langle M_1 \rangle \in REGULAR_{TM}$. If yes, *accept*. If no, *reject*."

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM* R decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_1:
     M_1="On input x:
     1. If $x = 0^n 1^n$, *accept*. (Recall $0^n 1^n$ is non-regular.)
     2. If $x \neq 0^n 1^n$, run M on w and *accept* if M accepts w."
  2. Use *TM* R to determine if $\langle M_1 \rangle \in REGULAR_{TM}$. If yes, *accept*. If no, *reject*."
- M_1 accepts the regular language Σ^* if $\langle M, w \rangle \in A_{TM}$, otherwise it only accepts the non-regular language $\{0^n 1^n\}$.

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM* R decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_1:
     M_1="On input x:
     1. If $x = 0^n 1^n$, *accept*. (Recall $0^n 1^n$ is non-regular.)
     2. If $x \neq 0^n 1^n$, run M on w and *accept* if M accepts w."
  2. Use *TM* R to determine if $\langle M_1 \rangle \in REGULAR_{TM}$. If yes, *accept*. If no, *reject*."
- M_1 accepts the regular language Σ^* if $\langle M, w \rangle \in A_{TM}$, otherwise it only accepts the non-regular language $\{0^n 1^n\}$.
- S is a decider for A_{TM}, this is a contradiction.

$REGULAR_{TM}$ is Undecidable

- $REGULAR_{TM} = \{\langle M \rangle \,|\, L(M)$ is a regular language$\}$
- Assume *TM* R decides $REGULAR_{TM}$.
- S="On input $\langle M, w \rangle$:
  1. Use $\langle M, w \rangle$ to create M_1:
     M_1="On input x:
     1. If $x = 0^n 1^n$, *accept*. (Recall $0^n 1^n$ is non-regular.)
     2. If $x \neq 0^n 1^n$, run M on w and *accept* if M accepts w."
  2. Use *TM* R to determine if $\langle M_1 \rangle \in REGULAR_{TM}$. If yes, *accept*. If no, *reject*."
- M_1 accepts the regular language Σ^* if $\langle M, w \rangle \in A_{TM}$, otherwise it only accepts the non-regular language $\{0^n 1^n\}$.
- S is a decider for A_{TM}, this is a contradiction.
- $\therefore$ $REGULAR_{TM}$ is undecidable.

EQ_{TM} is Undecidable

- $E_{TM} = \{\langle M \rangle \,|\, L(M) = \emptyset\}$, this is undecidable.

EQ_{TM} is Undecidable

- $E_{TM} = \{\langle M \rangle \,|\, L(M) = \emptyset\}$, this is undecidable.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$

EQ_{TM} is Undecidable

- $E_{TM} = \{\langle M \rangle \,|\, L(M) = \emptyset\}$, this is undecidable.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$
- Let M_2 be the TM that rejects everything.

EQ_{TM} is Undecidable

- $E_{TM} = \{\langle M \rangle \,|\, L(M) = \emptyset\}$, this is undecidable.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$
- Let M_2 be the TM that rejects everything.
- Define the mapping:

$$\langle M \rangle \mapsto \langle M, M_2 \rangle$$

from E_{TM} to EQ_{TM}. That is $L(M)$ is the empty language if and only if $L(M) = L(M_2)$.

EQ_{TM} is Undecidable

- $E_{TM} = \{\langle M \rangle \,|\, L(M) = \emptyset\}$, this is undecidable.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$
- Let M_2 be the TM that rejects everything.
- Define the mapping:

$$\langle M \rangle \mapsto \langle M, M_2 \rangle$$

  from E_{TM} to EQ_{TM}. That is $L(M)$ is the empty language if and only if $L(M) = L(M_2)$.
- EQ_{TM} decidable would imply E_{TM} is decidable, a contradiction.

EQ_{TM} is Undecidable

- $E_{TM} = \{\langle M \rangle \,|\, L(M) = \emptyset\}$, this is undecidable.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$
- Let M_2 be the TM that rejects everything.
- Define the mapping:

$$\langle M \rangle \mapsto \langle M, M_2 \rangle$$

  from E_{TM} to EQ_{TM}. That is $L(M)$ is the empty language if and only if $L(M) = L(M_2)$.
- EQ_{TM} decidable would imply E_{TM} is decidable, a contradiction.
- $\therefore$ EQ_{TM} is undecidable.

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid L(M_1) = L(M_2)\}$

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) \neq L(M_2)\}$

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) \neq L(M_2)\}$
- Claim $A_{TM} \leq_m \overline{EQ_{TM}}$ using the mapping:
  $F =$ "On input $\langle M, w \rangle$:

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2\rangle \,|\, L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle \,|\, L(M_1) \neq L(M_2)\}$
- Claim $A_{TM} \leq_m \overline{EQ_{TM}}$ using the mapping:
  F="On input $\langle M, w\rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Reject*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2\rangle \,|\, L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle \,|\, L(M_1) \neq L(M_2)\}$
- Claim $A_{TM} \leq_m \overline{EQ_{TM}}$ using the mapping:
  F="On input $\langle M, w\rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Reject*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2\rangle$."

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2\rangle \,|\, L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle \,|\, L(M_1) \neq L(M_2)\}$
- Claim $A_{TM} \leq_m \overline{EQ_{TM}}$ using the mapping:
  F="On input $\langle M, w\rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Reject*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2\rangle$."
- $L(M_1) \neq L(M_2)$ iff $\langle M, w\rangle \in A_{TM}$

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle \mid L(M_1) \neq L(M_2)\}$
- Claim $A_{TM} \leq_m \overline{EQ_{TM}}$ using the mapping:
  F="On input $\langle M, w\rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Reject*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2\rangle$."
- $L(M_1) \neq L(M_2)$ iff $\langle M, w\rangle \in A_{TM}$
- $\therefore \overline{A_{TM}} \leq_m EQ_{TM}$

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2\rangle \,|\, L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle \,|\, L(M_1) \neq L(M_2)\}$
- Claim $A_{TM} \leq_m \overline{EQ_{TM}}$ using the mapping:
  F="On input $\langle M, w\rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Reject*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2\rangle$."
- $L(M_1) \neq L(M_2)$ iff $\langle M, w\rangle \in A_{TM}$
- $\therefore \overline{A_{TM}} \leq_m EQ_{TM}$
- $\overline{A_{TM}}$ is not Turing-recognizable.

EQ_{TM} and Recognizability (Part 1)

- $EQ_{TM} = \{\langle M_1, M_2\rangle \,|\, L(M_1) = L(M_2)\}$
- $\overline{EQ_{TM}} = \{\langle M_1, M_2\rangle \,|\, L(M_1) \neq L(M_2)\}$
- Claim $A_{TM} \leq_m \overline{EQ_{TM}}$ using the mapping:
  F="On input $\langle M, w\rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Reject*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2\rangle$."
- $L(M_1) \neq L(M_2)$ iff $\langle M, w\rangle \in A_{TM}$
- $\therefore \overline{A_{TM}} \leq_m EQ_{TM}$
- $\overline{A_{TM}}$ is not Turing-recognizable.
- $\therefore EQ_{TM}$ is not Turing-recognizable.

EQ_{TM} and Recognizability (Part 2)

- Claim $A_{TM} \leq_m EQ_{TM}$ using the mapping:
  G="On input $\langle M, w \rangle$:

EQ_{TM} and Recognizability (Part 2)

- Claim $A_{TM} \leq_m EQ_{TM}$ using the mapping:
  $G =$ "On input $\langle M, w \rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Accept*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."

EQ_{TM} and Recognizability (Part 2)

- Claim $A_{TM} \leq_m EQ_{TM}$ using the mapping:
  $G =$ "On input $\langle M, w \rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Accept*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2 \rangle$."

EQ_{TM} and Recognizability (Part 2)

- Claim $A_{TM} \leq_m EQ_{TM}$ using the mapping:
  G="On input $\langle M, w \rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Accept*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2 \rangle$."
- $L(M_1) = L(M_2)$ iff $\langle M, w \rangle \in A_{TM}$

EQ_{TM} and Recognizability (Part 2)

- Claim $A_{TM} \leq_m EQ_{TM}$ using the mapping:
  G="On input $\langle M, w \rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Accept*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2 \rangle$."
- $L(M_1) = L(M_2)$ iff $\langle M, w \rangle \in A_{TM}$
- $\therefore \overline{A_{TM}} \leq_m \overline{EQ_{TM}}$

EQ_{TM} and Recognizability (Part 2)

- Claim $A_{TM} \leq_m EQ_{TM}$ using the mapping:
  G="On input $\langle M, w \rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Accept*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2 \rangle$."
- $L(M_1) = L(M_2)$ iff $\langle M, w \rangle \in A_{TM}$
- $\therefore \overline{A_{TM}} \leq_m \overline{EQ_{TM}}$
- $\overline{A_{TM}}$ is not Turing-recognizable.

EQ_{TM} and Recognizability (Part 2)

- Claim $A_{TM} \leq_m EQ_{TM}$ using the mapping:
  G="On input $\langle M, w \rangle$:
  1. Construct M_1 and M_2:

     $M_1 =$ "On any input: *Accept*"

     $M_2 =$ "On any input: Run M on w, *accept* if it accepts."
  2. Output $\langle M_1, M_2 \rangle$."
- $L(M_1) = L(M_2)$ iff $\langle M, w \rangle \in A_{TM}$
- $\therefore \overline{A_{TM}} \leq_m \overline{EQ_{TM}}$
- $\overline{A_{TM}}$ is not Turing-recognizable.
- $\therefore \overline{EQ_{TM}}$ is not Turing-recognizable.

EQ_{TM} and Recognizability

Theorem

The language $EQ_{TM} = \{\langle M_1, M_2 \rangle \,|\, L(M_1) = L(M_2)\}$ is neither Turing-recognizable nor co-Turing-recognizable.

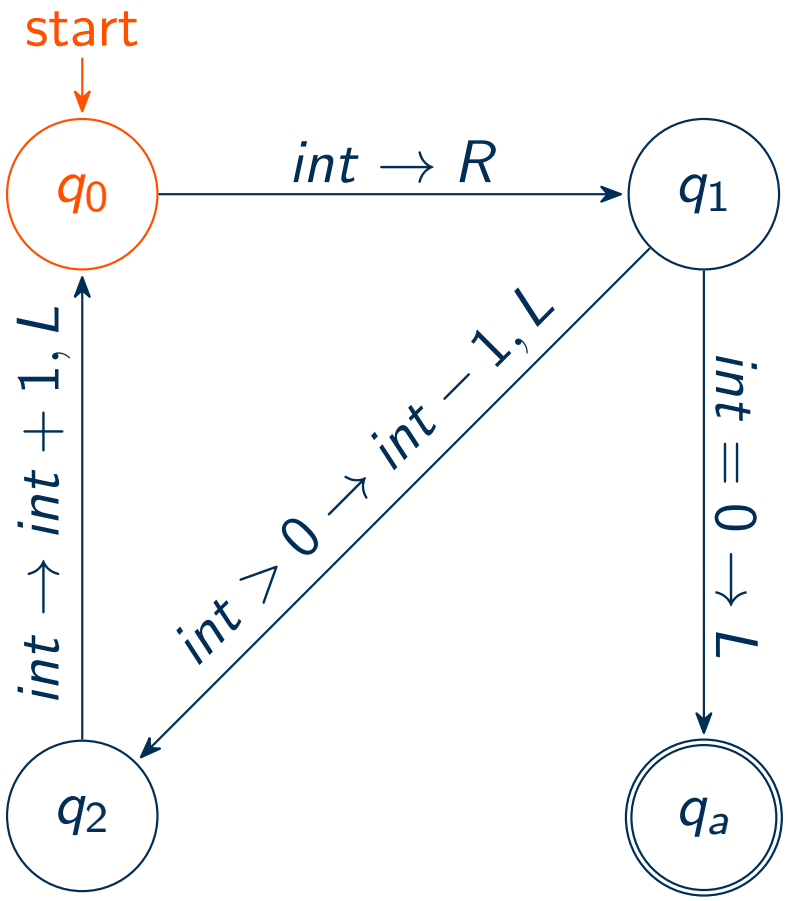
Table of Contents

Computational History

Definition (Computational History)

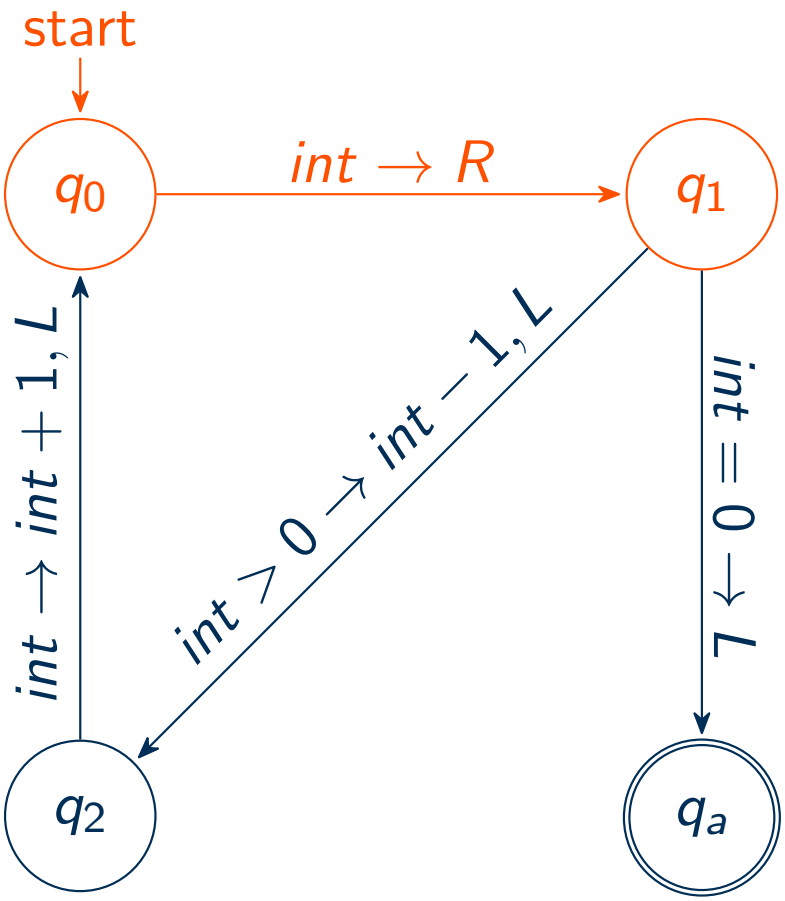
Let M be a Turing Machine and w an input string. An *accepting computational history* for M on w is a sequence of configurations, $C_1, C_2, C_3, \ldots, C_l$, where C_1 is the starting configuration, C_l is an accepting configuration, and each C_i follows from C_{i-1} according to the transition rules of M. A *rejecting computational history* is defined similarly, except C_l is a rejecting configuration.

Computational History



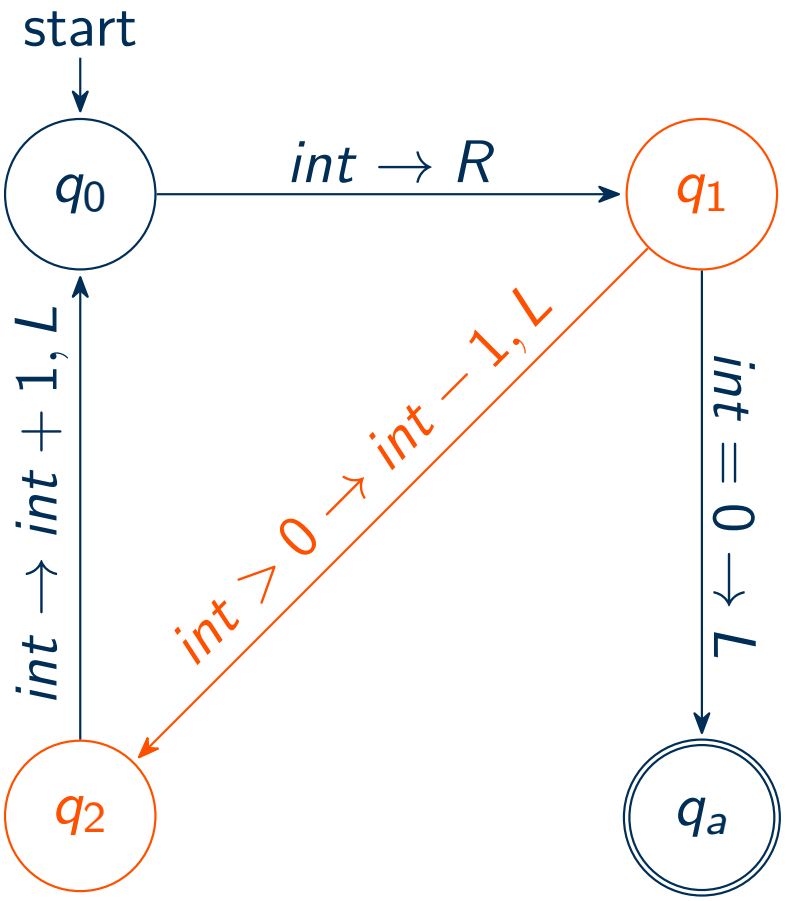
C_1: (q_0) 7 2

Computational History



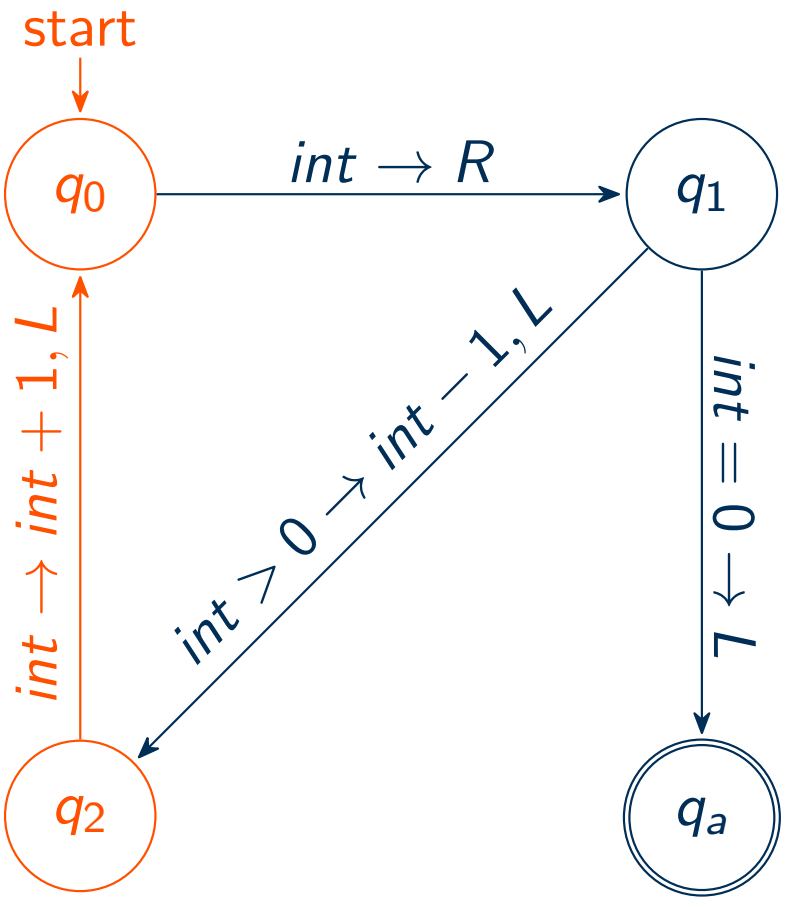
C_1:	(q_0)	7	2
C_2:	7	(q_1)	2

Computational History



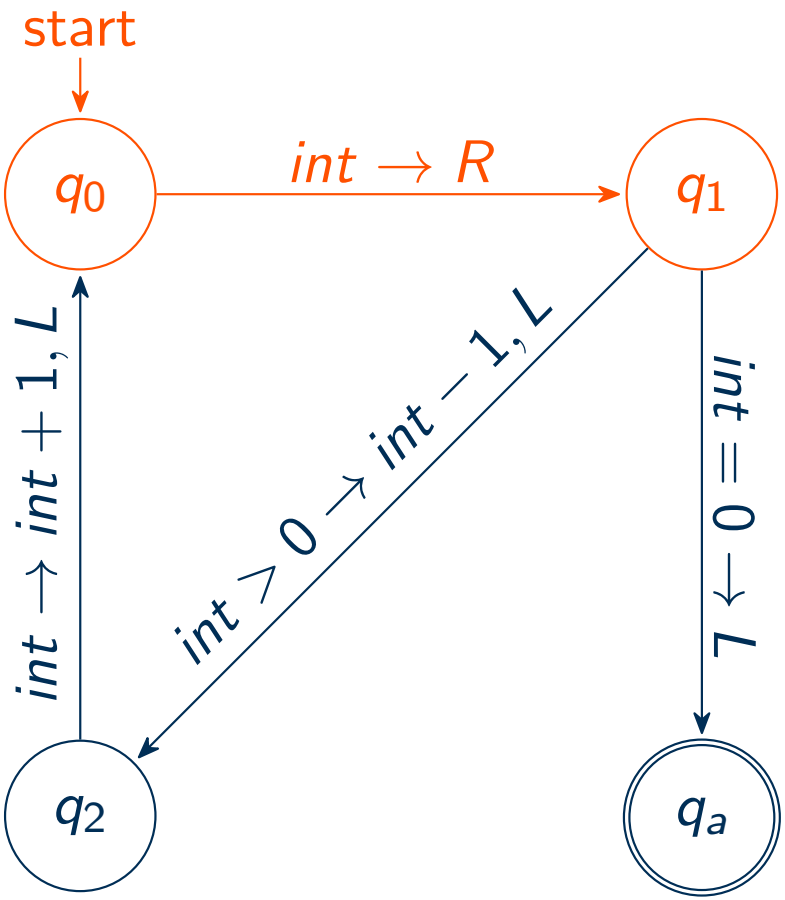
C_1:	(q_0)	7	2
C_2:	7	(q_1)	2
C_3:	(q_2)	7	1

Computational History



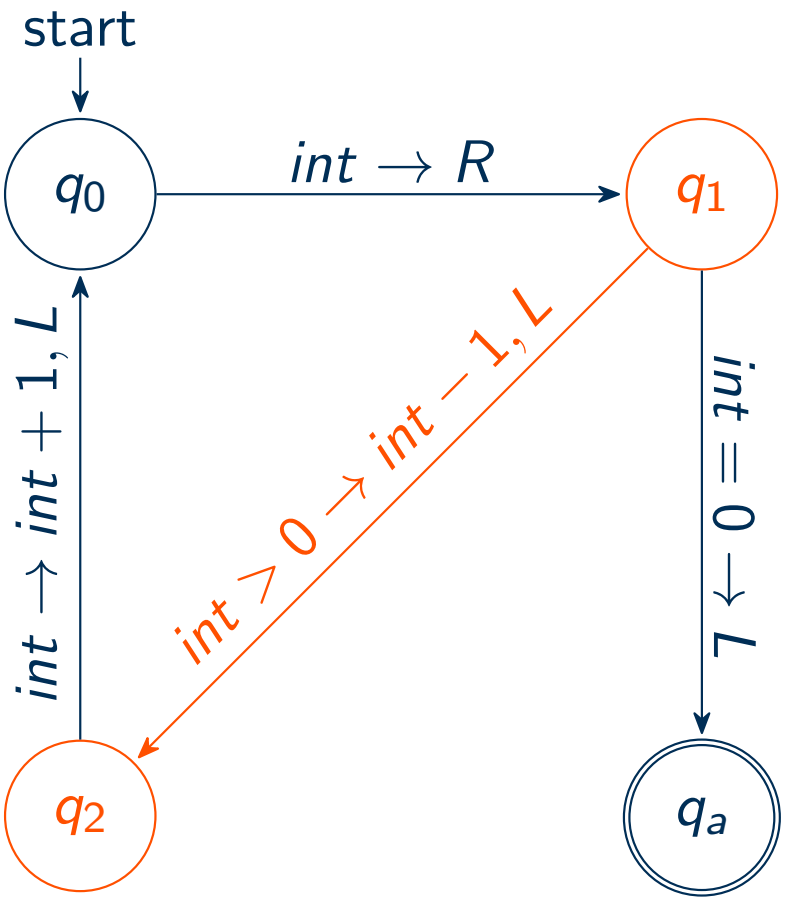
C_1:	(q_0)	7	2
C_2:	7	(q_1)	2
C_3:	(q_2)	7	1
C_4:	(q_0)	8	1

Computational History



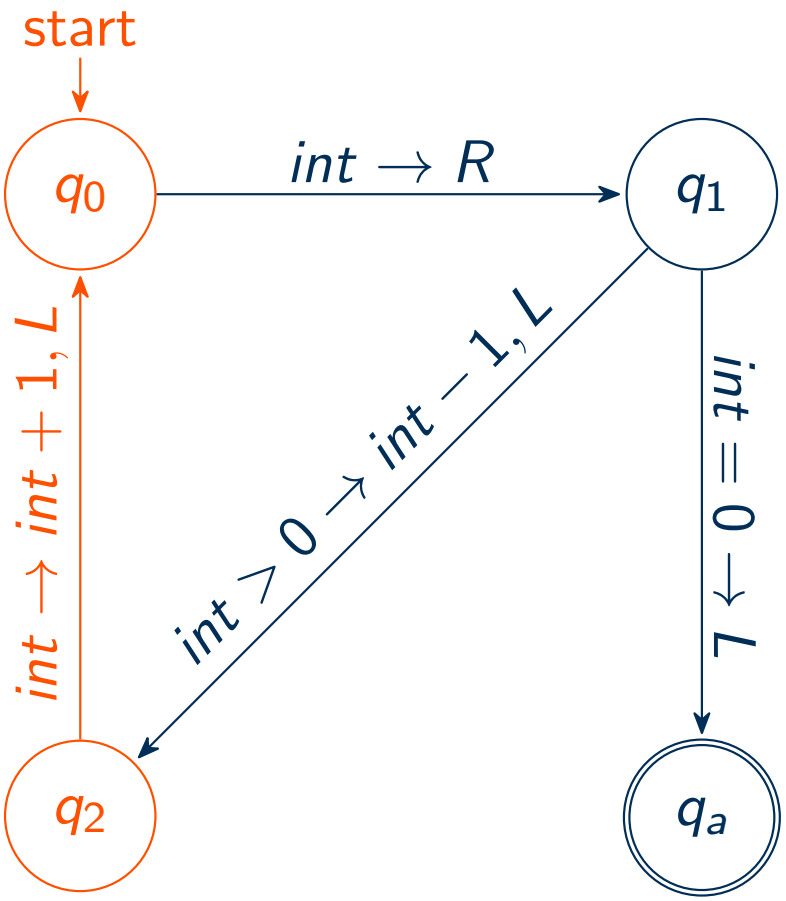
C_1:	(q_0)	7	2
C_2:	7	(q_1)	2
C_3:	(q_2)	7	1
C_4:	(q_0)	8	1
C_5:	8	(q_1)	1

Computational History



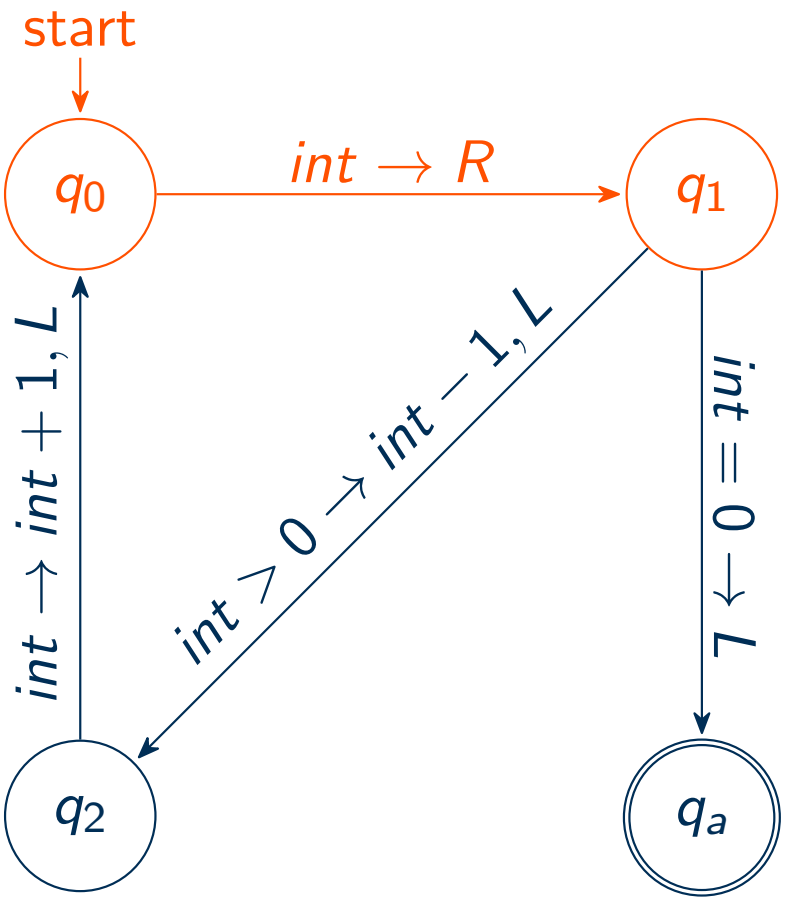
C_1:	(q_0)	7	2
C_2:	7	(q_1)	2
C_3:	(q_2)	7	1
C_4:	(q_0)	8	1
C_5:	8	(q_1)	1
C_6:	(q_2)	8	0

Computational History



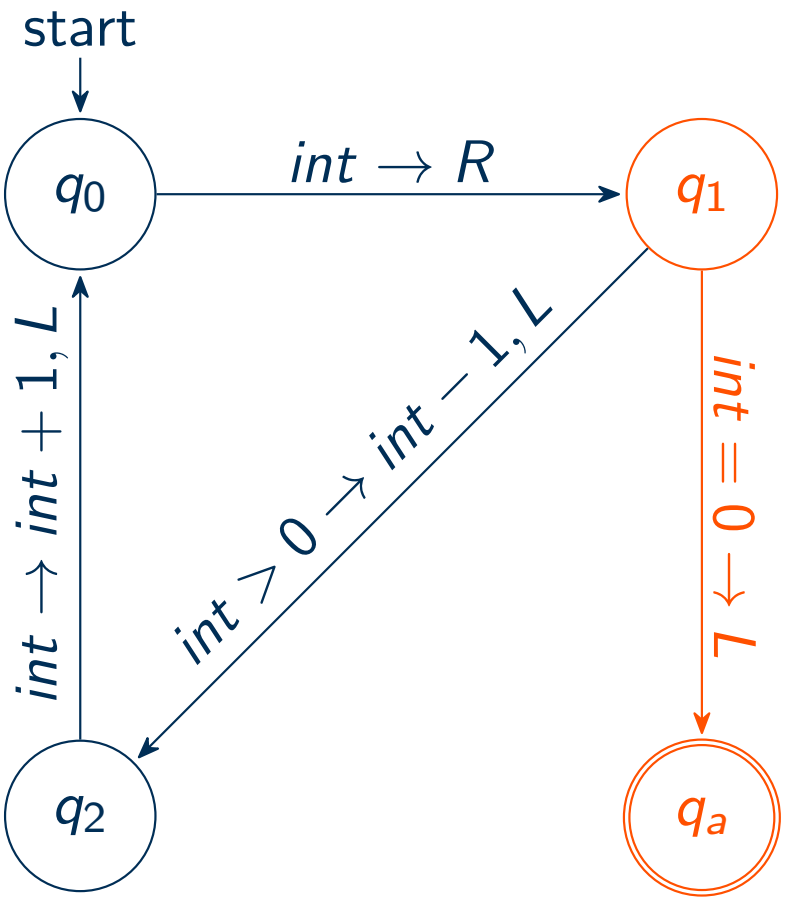
C_1:	(q_0)	7	2
C_2:	7	(q_1)	2
C_3:	(q_2)	7	1
C_4:	(q_0)	8	1
C_5:	8	(q_1)	1
C_6:	(q_2)	8	0
C_7:	(q_0)	9	0

Computational History



C_1:	(q_0)	7	2
C_2:	7	(q_1)	2
C_3:	(q_2)	7	1
C_4:	(q_0)	8	1
C_5:	8	(q_1)	1
C_6:	(q_2)	8	0
C_7:	(q_0)	9	0
C_8:	9	(q_1)	0

Computational History



C_1:	(q_0)	7	2
C_2:	7	(q_1)	2
C_3:	(q_2)	7	1
C_4:	(q_0)	8	1
C_5:	8	(q_1)	1
C_6:	(q_2)	8	0
C_7:	(q_0)	9	0
C_8:	9	(q_1)	0
C_9:	(q_a)	9	0

Linear Bounded Automaton

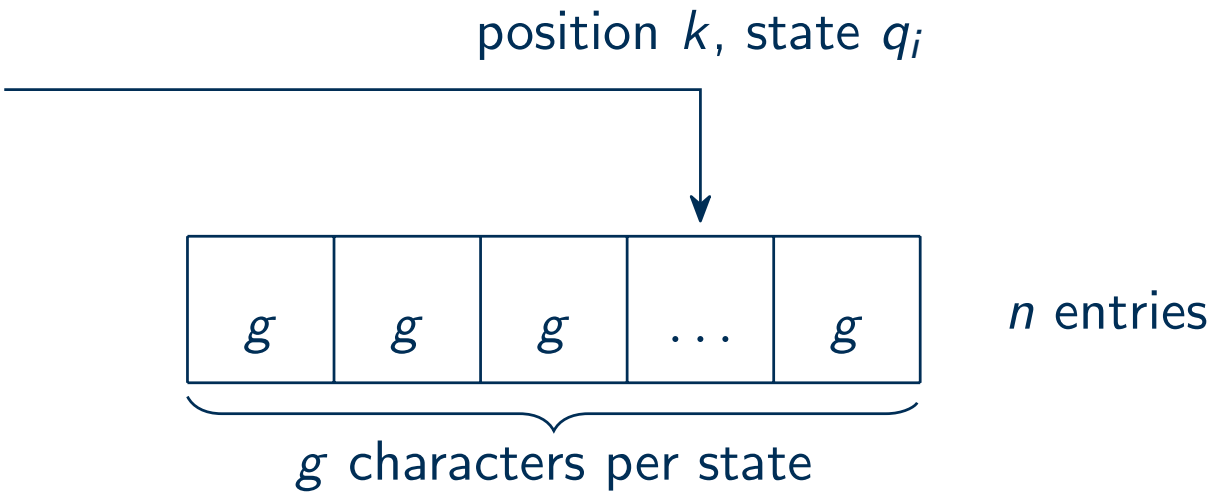
Definition

A *linear bounded automaton* is a Turing machine in which the head is restricted to the input tape and may not move off the left or right-hand ends of the tape.

Linear Bounded Automaton

Lemma

Let M be an LBA with q states and g symbols in the state alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.



A_{LBA} is Decidable

- $A_{LBA} = \{\langle M, w\rangle \,|\, M \text{ is an LBA that accepts } w\}$

A_{LBA} is Decidable

- $A_{LBA} = \{\langle M, w\rangle \,|\, M$ is an LBA that accepts $w\}$
- Any given LBA M has at most qng^n distinct states.

A_{LBA} is Decidable

- $A_{LBA} = \{\langle M, w\rangle \,|\, M$ is an LBA that accepts $w\}$
- Any given LBA M has at most qng^n distinct states.
- Simulate M on w for at most qng^n steps:

A_{LBA} is Decidable

- $A_{LBA} = \{\langle M, w\rangle \,|\, M$ is an LBA that accepts $w\}$
- Any given LBA M has at most qng^n distinct states.
- Simulate M on w for at most qng^n steps:
  1. If it accepts, *accept*.

A_{LBA} is Decidable

- $A_{LBA} = \{\langle M, w\rangle \,|\, M$ is an LBA that accepts $w\}$
- Any given LBA M has at most qng^n distinct states.
- Simulate M on w for at most qng^n steps:
  1. If it accepts, *accept*.
  2. If it rejects, *reject*.

A_{LBA} is Decidable

- $A_{LBA} = \{\langle M, w\rangle \,|\, M$ is an LBA that accepts $w\}$
- Any given LBA M has at most qng^n distinct states.
- Simulate M on w for at most qng^n steps:
  1. If it accepts, *accept*.
  2. If it rejects, *reject*.
  3. If it hasn't yet halted, *reject*.

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B \text{ is a LBA and } L(B) = \emptyset \}$

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B \text{ is a LBA and } L(B) = \emptyset \}$
2. Let M be a TM and w a string

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B \text{ is a LBA and } L(B) = \emptyset\ \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

a finite sequence of configurations; B accepts x if

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B \text{ is a LBA and } L(B) = \emptyset\ \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

a finite sequence of configurations; B accepts x if

1. $C_1 = (q_0)w$, the starting configuration for M simulating w

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B \text{ is a LBA and } L(B) = \emptyset\ \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

a finite sequence of configurations; B accepts x if

1. $C_1 = (q_0)w$, the starting configuration for M simulating w
2. C_l is an accepting configuration in M

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B \text{ is a LBA and } L(B) = \emptyset\ \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

a finite sequence of configurations; B accepts x if

1. $C_1 = (q_0)w$, the starting configuration for M simulating w
2. C_l is an accepting configuration in M
3. C_{i+1} follows from C_i using the rules of M

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B \text{ is a LBA and } L(B) = \emptyset\ \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

a finite sequence of configurations; B accepts x if

   1. $C_1 = (q_0)w$, the starting configuration for M simulating w
   2. C_l is an accepting configuration in M
   3. C_{i+1} follows from C_i using the rules of M

4. Suppose R decides emptiness for LBA

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B$ is a LBA and $L(B) = \emptyset \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

   a finite sequence of configurations; B accepts x if
   1. $C_1 = (q_0)w$, the starting configuration for M simulating w
   2. C_l is an accepting configuration in M
   3. C_{i+1} follows from C_i using the rules of M
4. Suppose R decides emptiness for LBA
5. If R rejects $\langle B \rangle$, *accept*

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B$ is a LBA and $L(B) = \emptyset \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

   a finite sequence of configurations; B accepts x if
   1. $C_1 = (q_0)w$, the starting configuration for M simulating w
   2. C_l is an accepting configuration in M
   3. C_{i+1} follows from C_i using the rules of M
4. Suppose R decides emptiness for LBA
5. If R rejects $\langle B \rangle$, *accept*
6. If R accepts $\langle B \rangle$, *reject*

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B$ is a LBA and $L(B) = \emptyset \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

   a finite sequence of configurations; B accepts x if
   1. $C_1 = (q_0)w$, the starting configuration for M simulating w
   2. C_l is an accepting configuration in M
   3. C_{i+1} follows from C_i using the rules of M
4. Suppose R decides emptiness for LBA
5. If R rejects $\langle B \rangle$, *accept*
6. If R accepts $\langle B \rangle$, *reject*
7. $\langle M, w \rangle \in A_{TM}$ is decidable, a contradiction

E_{LBA} is Undecidable

1. $E_{LBA} = \{\langle B \rangle \mid B$ is a LBA and $L(B) = \emptyset \}$
2. Let M be a TM and w a string
3. Construct an LBA B to read

$$x = \#C_1\#C_2\#C_3\#\cdots\#C_l\#$$

   a finite sequence of configurations; B accepts x if
   1. $C_1 = (q_0)w$, the starting configuration for M simulating w
   2. C_l is an accepting configuration in M
   3. C_{i+1} follows from C_i using the rules of M
4. Suppose R decides emptiness for LBA
5. If R rejects $\langle B \rangle$, *accept*
6. If R accepts $\langle B \rangle$, *reject*
7. $\langle M, w \rangle \in A_{TM}$ is decidable, a contradiction
8. $\therefore E_{LBA}$ is undecidable

ALL_{CFG} is Undecidable

1. $All_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \Sigma^*\}$

ALL_{CFG} is Undecidable

1. $All_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \Sigma^*\}$
2. Let M be a TM and w a string

ALL_{CFG} is Undecidable

1. $All_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \Sigma^*\}$
2. Let M be a TM and w a string
3. Design a PDA D that reads computation histories of the form

$$x = \#C_1\#C_2^{\mathcal{R}}\#C_3\#C_4^{\mathcal{R}}\#\cdots\#C_l\#$$

and accepts if x isn't a history of M accepting w.

ALL_{CFG} is Undecidable

1. $All_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \Sigma^*\}$
2. Let M be a TM and w a string
3. Design a PDA D that reads computation histories of the form

$$x = \#C_1\#C_2^{\mathcal{R}}\#C_3\#C_4^{\mathcal{R}}\#\cdots\#C_l\#$$

   and accepts if x isn't a history of M accepting w.
4. Suppose R decides ALL_{CFG} and apply it to "D"

ALL_{CFG} is Undecidable

1. $All_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \Sigma^*\}$
2. Let M be a TM and w a string
3. Design a PDA D that reads computation histories of the form

$$x = \#C_1\#C_2^{\mathcal{R}}\#C_3\#C_4^{\mathcal{R}}\#\cdots\#C_l\#$$

   and accepts if x isn't a history of M accepting w.
4. Suppose R decides ALL_{CFG} and apply it to "D"
5. $L(D) = \Sigma^*$ iff M doesn't accept w

ALL_{CFG} is Undecidable

1. $All_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \Sigma^*\}$
2. Let M be a TM and w a string
3. Design a PDA D that reads computation histories of the form

$$x = \#C_1\#C_2^{\mathcal{R}}\#C_3\#C_4^{\mathcal{R}}\#\cdots\#C_l\#$$

   and accepts if x isn't a history of M accepting w.
4. Suppose R decides ALL_{CFG} and apply it to "D"
5. $L(D) = \Sigma^*$ iff M doesn't accept w
6. $\langle M, w \rangle \in A_{TM}$ is decidable, a contradiction

ALL_{CFG} is Undecidable

1. $All_{CFG} = \{\langle G \rangle \,|\, G \text{ is a CFG and } L(G) = \Sigma^*\}$
2. Let M be a TM and w a string
3. Design a PDA D that reads computation histories of the form

$$x = \#C_1\#C_2^{\mathcal{R}}\#C_3\#C_4^{\mathcal{R}}\#\cdots\#C_l\#$$

   and accepts if x isn't a history of M accepting w.
4. Suppose R decides ALL_{CFG} and apply it to "D"
5. $L(D) = \Sigma^*$ iff M doesn't accept w
6. $\langle M, w \rangle \in A_{TM}$ is decidable, a contradiction
7. $\therefore$ ALL_{CFG} is undecidable

CFLs

Theorem

The set

$$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

is not a decidable language. (Proof held until after Chapter 5.)

Table of Contents

Next Class

- Recursion Theorem

Next Class

- Recursion Theorem
- Another proof that A_{TM} is undecidable (sort of)

Next Class

- Recursion Theorem
- Another proof that A_{TM} is undecidable (sort of)
- Decidability and Logical Theories

Limits of Turing Machines (Part 2)

Dr. Chuck Rocca
roccac@wcsu.edu
http://sites.wcsu.edu/roccac

WESTERN CONNECTICUT STATE UNIVERSITY
MACRICOSTAS SCHOOL OF ARTS & SCIENCES