King Saud University

Theory of Computation

CSC 339 – Spring 2021

Chapter-5: part1

Reducability

King Saud University

Department of Computer Science

Dr. Azzam Alsudais

Introduction

- **Some problems can be reduced to other new problems.**

Introduction

- **Some problems can be reduced to other <u>new</u> problems.**

- **This reducability helps us reason about the decidability of certain problems.**

Introduction

- **Some problems can be reduced to other <u>new</u> problems.**

- **This reducability helps us reason about the decidability of certain problems.**

- **We say problem A is reducible to problem B:**

Introduction

- **Some problems can be reduced to other <u>new</u> problems.**

- **This reducability helps us reason about the decidability of certain problems.**

- **We say problem A is reducible to problem B:**
  - **Solving A cannot be harder than solving B because a solution to B gives a solution to A.**

Introduction

- **Some problems can be reduced to other <u>new</u> problems.**

- **This reducability helps us reason about the decidability of certain problems.**

- **We say problem A is reducible to problem B:**
  - **Solving A cannot be harder than solving B because a solution to B gives a solution to A.**
  - **If A is reducible to B, and B is decidable, then A also is decidable.**

Introduction

- **Some problems can be reduced to other new problems.**

- **This reducability helps us reason about the decidability of certain problems.**

- **We say problem A is reducible to problem B:**
  - **Solving A cannot be harder than solving B because a solution to B gives a solution to A.**
  - **If A is reducible to B, and B is decidable, then A also is decidable.**
  - **If A is undecidable, then B is undecidable, too.**

Introduction

- **Some problems can be reduced to other <u>new</u> problems.**

- **This reducability helps us reason about the decidability of certain problems.**

- **We say problem A is reducible to problem B:**

Goal is to prove that a given problem B is undecidable by showing that some other problem that's known to be undecidable A reduces to it.

The Halting Problem

- **The halting problem** $HALT_{TM}$ **determines whether a given TM halts (accept or reject) on some input string** w**.**

The Halting Problem

- **The halting problem** $HALT_{TM}$ **determines whether a given TM halts (accept or reject) on some input string** w**.**

- A_{TM} **on the other hand determines whether a given TM accepts a given string** w**.**

The Halting Problem

- **The halting problem** $HALT_{TM}$ **determines whether a given TM halts (accept or reject) on some input string** w**.**

- A_{TM} **on the other hand determines whether a given TM accepts a given string** w**.**

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}.$$

The Halting Problem

- **The halting problem** $HALT_{TM}$ **determines whether a given TM halts (accept or reject) on some input string** w**.**

- A_{TM} **on the other hand determines whether a given TM accepts a given string** w**.**

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}.$$

Theorem 5.1

$HALT_{TM}$ is undecidable

The Halting Problem: Undecidability Proof

- **To prove that $HALT_{TM}$ is undecidable, we reduce A_{TM} to it.**

The Halting Problem: Undecidability Proof

- **To prove that $HALT_{TM}$ is undecidable, we reduce A_{TM} to it.**
- **Proof by contradiction**
  - **Assume $HALT_{TM}$ is decidable**
  - **Reduce A_{TM} to $HALT_{TM}$**
  - **Show that since A_{TM} is undecidable and that it reduces to $HALT_{TM}$, then $HALT_{TM}$ is undecidable.**

The Halting Problem: Undecidability Proof

- **Assume** $HALT_{TM}$ **is decidable by TM** R**.**
- **Construct TM** S **to decide** A_{TM}**.**

The Halting Problem: Undecidability Proof

- **Assume** $HALT_{TM}$ **is decidable by TM** R**.**

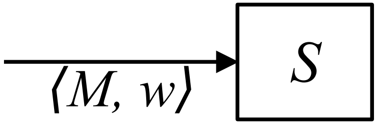
- **Construct TM** S **to decide** A_{TM}**.**

S = "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

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 has accepted, *accept*; if M has rejected, *reject*."

The Halting Problem: Undecidability Proof

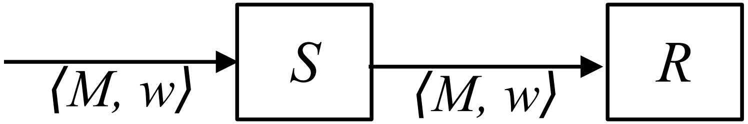
S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:
1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, reject.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, accept; if M has rejected, reject."



The Halting Problem: Undecidability Proof

S = "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

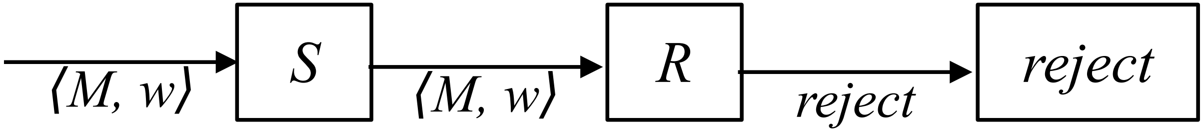
1. Run TM R on input $\langle M, w \rangle$.
2. If R rejects, reject.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, accept; if M has rejected, reject."



The Halting Problem: Undecidability Proof

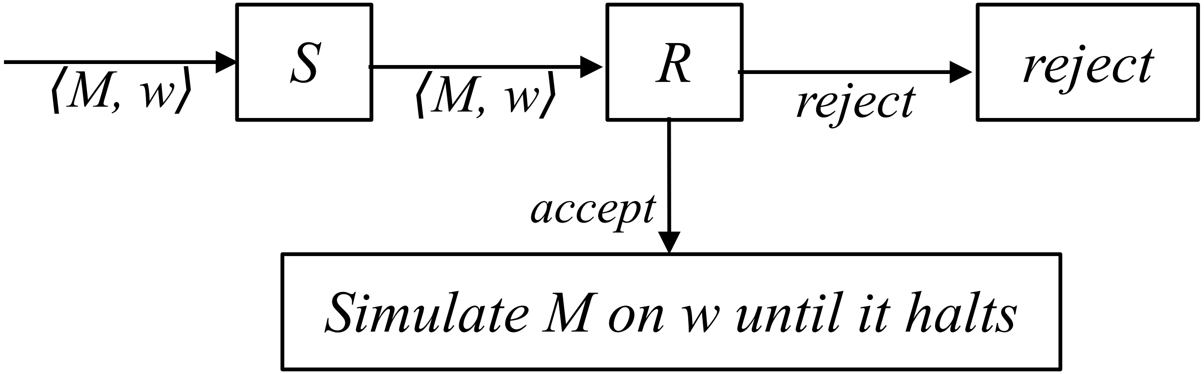
S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

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 has accepted, accept; if M has rejected, reject."



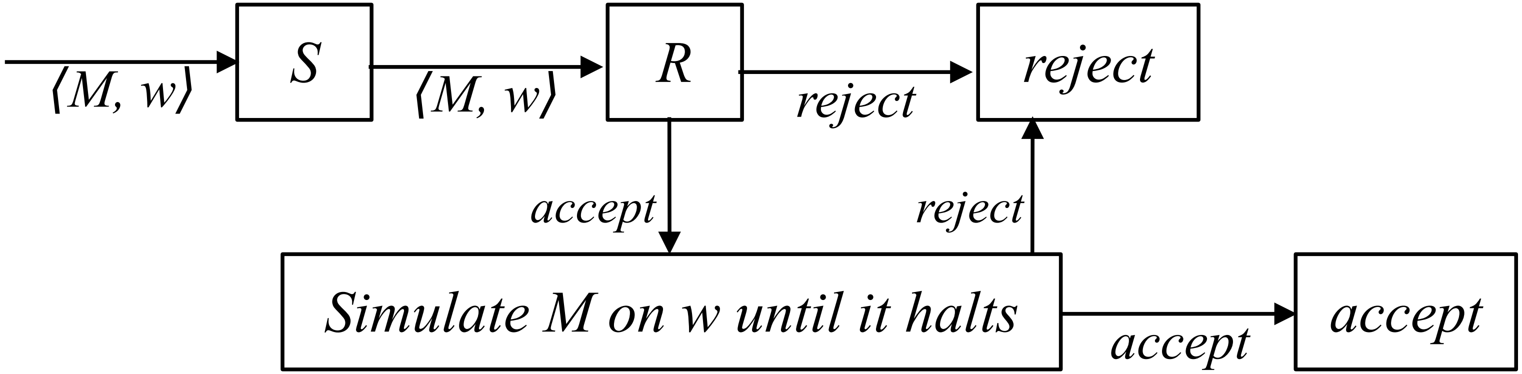
The Halting Problem: Undecidability Proof

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:
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 has accepted, accept; if M has rejected, reject."



The Halting Problem: Undecidability Proof

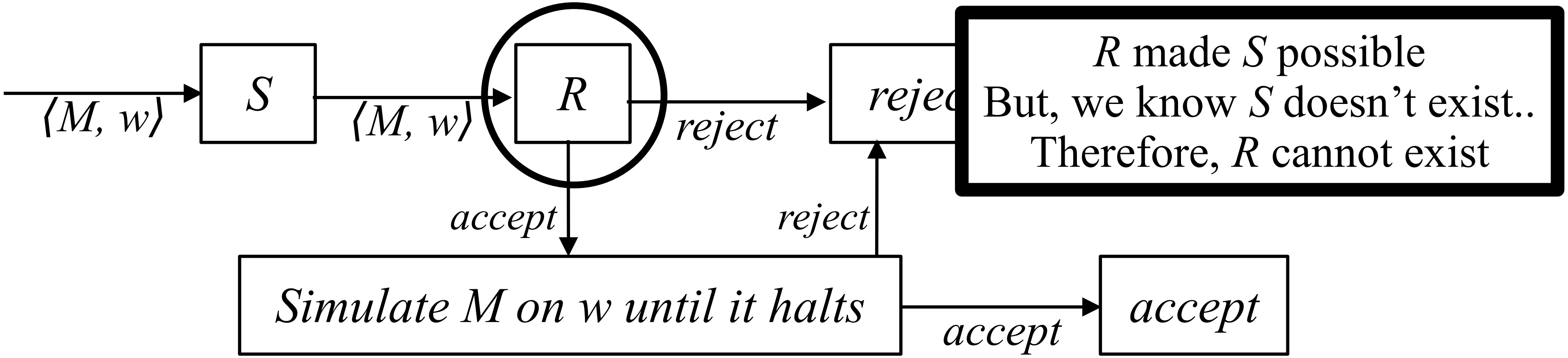
S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:
1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, reject.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, accept; if M has rejected, reject."



The Halting Problem: Undecidability Proof

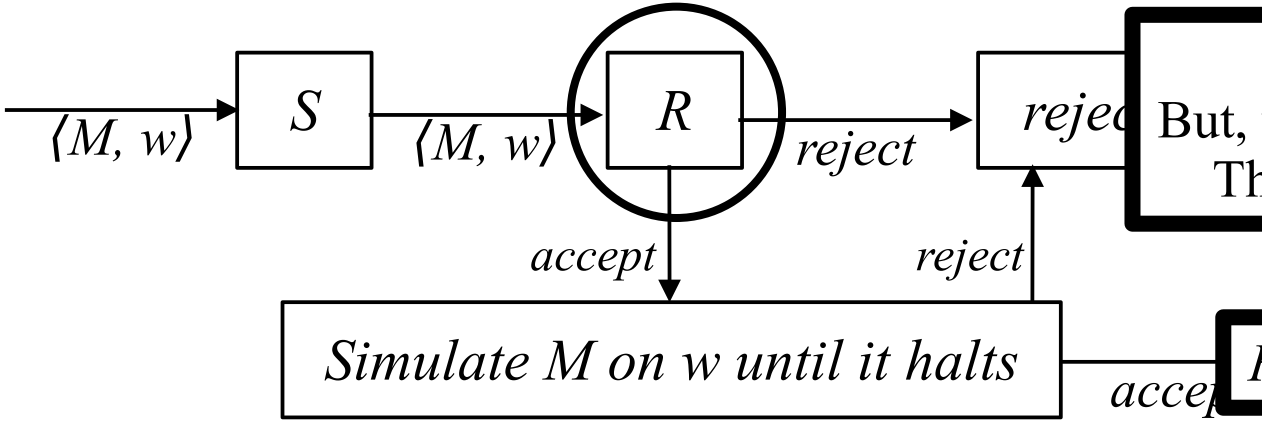
S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

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 has accepted, accept; if M has rejected, reject."



The Halting Problem: Undecidability Proof

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:
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 has accepted, accept; if M has rejected, reject."



R made S possible
But, we know S doesn't exist..
Therefore, R cannot exist

$HALT_{TM}$ is undecidable

The Emptiness Problem

- **The Emptiness problem E_{TM} is concerned with determining TMs whose language is empty.**

The Emptiness Problem

- **The Emptiness problem** E_{TM} **is concerned with determining TMs whose language is empty.**

- **This means we want to see the behavior of those TMs on all possible strings.**

The Emptiness Problem

- **The Emptiness problem E_{TM} is concerned with determining TMs whose language is empty.**
- **This means we want to see the behavior of those TMs on all possible strings.**
- **Given a description of a TM, we want to check if this TM does not accept any string.**

The Emptiness Problem

- **The Emptiness problem E_{TM} is concerned with determining TMs whose language is empty.**

- **This means we want to see the behavior of those TMs on all possible strings.**

- **Given a description of a TM, we want to check if this TM does not accept any string.**

- **A_{TM} and $HALT_{TM}$ are for determining the behavior of a given TM on a given string.**

The Emptiness Problem

- **The Emptiness problem E_{TM} is concerned with determining TMs whose language is empty.**
- **This means we want to see the behavior of those TMs on all possible strings.**
- **Given a description of a TM, we want to check if this TM does not accept any string.**
- **A_{TM} and $HALT_{TM}$ are for determining the behavior of a given TM on a given string.**
- **E_{TM}, on the other hand, concerns the behavior of a TM on all strings.**

The Emptiness Problem

$$\boxed{E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.}$$

The Emptiness Problem

$$\boxed{E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.}$$

If M does not accept any string, then we should ***accept*** M as a member of this language

The Emptiness Problem

$$\boxed{E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.}$$

If M does not accept any string, then we should ***accept*** M as a member of this language

If M accepts at least 1string, then we should ***reject*** that M is a member of this language

The Emptiness Problem

$$\boxed{E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.}$$

$$\boxed{E_{TM}} \xrightarrow{\text{Harder than}} \boxed{HALT_{TM}} \xrightarrow{\text{Harder than}} \boxed{A_{TM}}$$

The Emptiness Problem

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.$$

The idea here is to assume there is a decider R for E_{TM} and use R to build a decider S A_{TM}.

The Emptiness Problem: Undecidability Proof

- **How can we construct a decider** R **for** E_{TM} **and use it to decide** A_{TM}**?**

The Emptiness Problem: Undecidability Proof

- **How can we construct a decider R for E_{TM} and use it to decide A_{TM}?**

- **One idea when S receives $\langle M,w \rangle$**
  - **Run R on $\langle M \rangle$**
    - **If it accepts (meaning $L(M)$ is empty), then S rejects w.**
    - **If it rejects (meaning $L(M)$ is non-empty), then all we know is that $L(M)$ is not empty.**
      - **We cannot tell anything about string w.**

The Emptiness Problem: Undecidability Proof

- **How can we construct a decider** R **for** E_{TM} **and use it to decide** A_{TM}**?**

- **One idea when** S **receives** $\langle M,w \rangle$
  - **Run** R **on** $\langle M \rangle$
    - **If it accepts (meaning** $L(M)$ **is empty), then** S **rejects** w**.**
    - **If it rejects (meaning** $L(M)$ **is non-empty), then all we know is that** $L(M)$ **is not empty.**
      - **We cannot tell anything about string** w**.**
- **We need a different idea.**

The Emptiness Problem: Undecidability Proof

- **Alternatively, we can use the following idea.**

- **Instead of running R on $\langle M \rangle$, run R on a modified version of $\langle M \rangle$, which we will call M_1.**
- **What does M_1 do?**
  - **Rejects all strings excepts w.**
  - **If it sees w, then it runs the original TM M.**

The Emptiness Problem: Undecidability Proof

- **Alternatively, we can use the following idea.**
- **Instead of running *R* on ⟨*M*⟩, run *R* on a modified version of ⟨*M*⟩, which we will call M_1.**
- **What does M_1 do?**
  - **Rejects all strings excepts *w*.**
  - **If it sees *w*, then it runs the original TM *M*.**

M_1 = “On input x:
1. If $x \neq w$, reject .
2. If $x = w$, run M on input w and accept if M does.”

The Emptiness Problem: Undecidability Proof

M_1 = "On input x:

1. If $x \neq w$, reject .
2. If $x = w$, run *M* on input w and accept if *M* does."

The Emptiness Problem: Undecidability Proof

M_1 = "On input x:
1. If $x \neq w$, reject .
2. If $x = w$, run *M* on input w and accept if *M* does."

$$L(M_1) = \begin{cases} \varnothing, \textit{ if } M \textit{ does not accept } w \\ \{w\}, \textit{ if } M \textit{ accepts } w \end{cases}$$

The Emptiness Problem: Undecidability Proof

M_1 = "On input x:
1. If $x \neq w$, reject .
2. If $x = w$, run M on input w and accept if M does."

$$L(M_1) = \begin{cases} \varnothing, \text{ if } M \text{ does not accept } w \\ \{w\}, \text{ if } M \text{ accepts } w \end{cases}$$

Answering the question whether M accepts w is fundamental to answering whether the $L(M_1)$ is empty or not

The Emptiness Problem: Undecidability Proof

- **Now, we are ready to construct a decider *S* for A_{TM}.**

S = "On input ⟨M, w⟩, an encoding of a TM M and a string *w*:

1. Use the description of M and *w* to construct the TM M_1 just described.
2. Run R on input ⟨M_1⟩.
3. If R accepts, reject; if R rejects, accept."

The Emptiness Problem: Undecidability Proof

- **Now, we are ready to construct a decider *S* for A_{TM}.**

S = "On input ⟨M, w⟩, an encoding of a TM M and a string *w*:
1. Use the description of M and *w* to construct the TM M_1 just described.
2. Run R on input ⟨M_1⟩.
3. If R accepts, reject; if R rejects, accept."

S will accept ⟨M, w⟩ *iff* L(M_1) is non-empty

The Emptiness Problem: Undecidability Proof

- **Now, we are ready to construct a decider S for A_{TM}.**

S = "On input $\langle M, w \rangle$, an encoding of a TM M and a string *w*:

1. Use the description of M and *w* to construct the TM M_1 just described.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, reject; if R rejects, accept."

S will accept $\langle M, w \rangle$ *iff* $L(M_1)$ is non-empty

Contradiction!

Decider *S* in this case decides whether *w* is accepted by TM *M*, which gives the language A_{TM}. But, we know A_{TM} is undecidable. So, *S* doesn't exist. Therefore, *R* cannot exist.

Computation Histories

- **Definition:**

  Let M be a Turing machine and w an input string. An accepting computation history for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_k$, where C_1 is the start configuration of M on w, C_k is an accepting configuration of M, and each C_i legally follows from C_{i-1} according to the rules of M. A rejecting computation history for M on w is defined similarly, except that C_k is a rejecting configuration

Computation Histories

- **Definition:**

  Let M be a Turing machine and w an input string. An accepting computation history for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_k$, where C_1 is the start configuration of M on w, C_k is an accepting configuration of M, and each C_i legally follows from C_{i-1} according to the rules of M. A rejecting computation history for M on w is defined similarly, except that C_k is a rejecting configuration

- **Computation histories are finite.**

Computation Histories

- **Definition:**

  Let M be a Turing machine and w an input string. An accepting computation history for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_k$, where C_1 is the start configuration of M on w, C_k is an accepting configuration of M, and each C_i legally follows from C_{i-1} according to the rules of M. A rejecting computation history for M on w is defined similarly, except that C_k is a rejecting configuration

- **Computation histories are finite.**
- **Deterministic TMs have exactly one history for each input.**

Linearly Bounded Automaton (LBA)

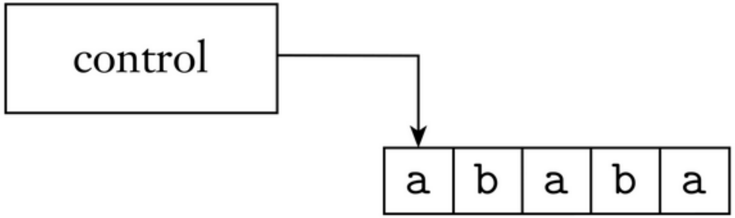
- **Definition:**

  A ***linear bounded automaton*** is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.

Linearly Bounded Automaton (LBA)

- **Definition:**

  A ***linear bounded automaton*** is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.



Linearly Bounded Automaton (LBA)

- **Definition:**

  A ***linear bounded automaton*** is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.

- **LBAs have limited power, yet they are still powerful.**

- **LBAs can decide** A_{DFA}, A_{CFG}, E_{DFA}, E_{CFG}.

Linearly Bounded Automaton (LBA)

- **Lemma 5.8:**

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

Linearly Bounded Automaton (LBA)

- **Lemma 5.8:**

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

- **A_{LBA} = {⟨M, w⟩ | M is an LBA that accepts string w}**

Linearly Bounded Automaton (LBA)

- **Lemma 5.8:**

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

- **A_{LBA} = {⟨M, w⟩ | M is an LBA that accepts string w}**

Theorem 5.9

A_{LBA} is decidable

A_{LBA} is decidable: Proof

- **Proof Idea**
  - **Simulate *M* on input *w*.**

A_{LBA} is decidable: Proof

- **Proof Idea**
  - **Simulate *M* on input *w*.**
  - **If *M* halts and accepts or rejects, we accept or reject accordingly.**

A_{LBA} is decidable: Proof

- **Proof Idea**
  - **Simulate *M* on input *w*.**
  - **If *M* halts and accepts or rejects, we accept or reject accordingly.**
  - **What if *M* loops on *w*? How can we tell whether *w* would be accepted?**

A_{LBA} is decidable: Proof

- **Proof Idea**
  - **Simulate *M* on input *w*.**
  - **If *M* halts and accepts or rejects, we accept or reject accordingly.**
  - **What if *M* loops on *w*? How can we tell whether *w* would be accepted?**
  - **Leverage lemma 5.8 (the amount of tape available to an LBA is limited).**

A_{LBA} is decidable: Proof

- **Proof Idea**
  - **Simulate *M* on input *w*.**
  - **If *M* halts and accepts or rejects, we accept or reject accordingly.**
  - **What if *M* loops on *w*? How can we tell whether *w* would be accepted?**
  - **Leverage lemma 5.8 (the amount of tape available to an LBA is limited).**
  - **If we exceed the number of distinct configurations (qng^n), then we know M is repeating some configurations. Thus, we reject.**

A_{LBA} is decidable: Proof

L = "On input ⟨M, w⟩, where M is an LBA and w is a string:

1. Simulate M on w for qng^n steps <u>or</u> until it halts.
2. If M has halted, *accept* if it has accepted and *reject* if it has rejected. If it has not halted, *reject*."

A_{LBA} is decidable: Proof

L = "On input ⟨M, w⟩, where M is an LBA and w is a string:

1. Simulate M on w for ***qng^n*** steps <u>or</u> until it halts.
2. If M has halted, *accept* if it has accepted and *reject* if it has rejected. If it has not halted, *reject*."

Meaning that it is repeating configurations (looping)