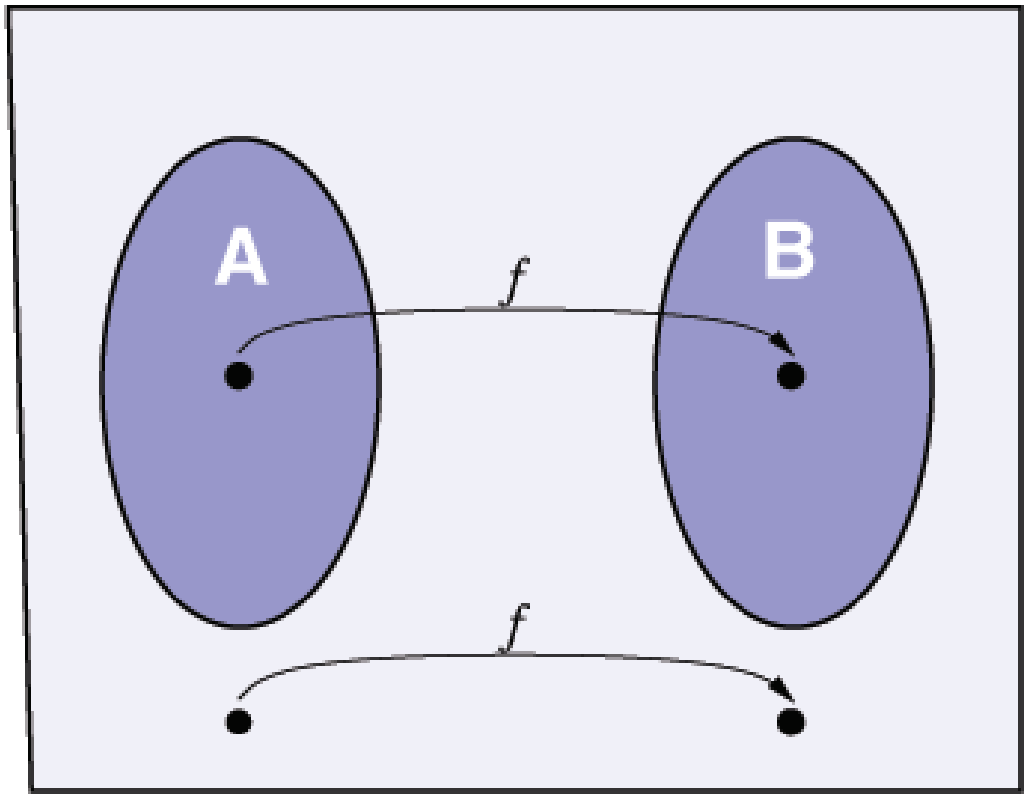
Computational Models: Lecture 9, Spring 2011

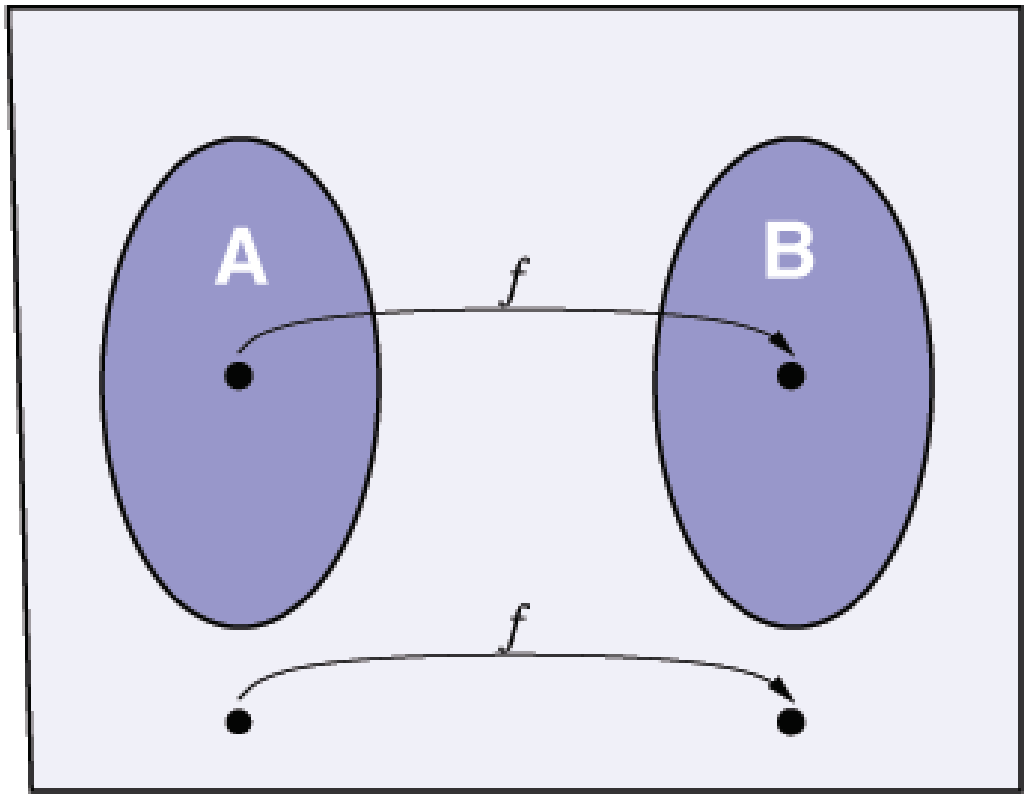
- Mapping Reductions
- Undecidability by Rice Theorem
- $\mathcal{RE}$-Completeness
- Reductions by computational histories

- Sipser's book, Chapter 5, Sections 5.1, 5.3

Mapping Reductions



Mapping Reductions



A mapping reduction converts questions about membership in A to membership in B

Mapping Reductions

Definition: Let A and B be two languages. We say that there is a mapping reduction from A to B, and denote

$$A \leq_m B$$

Mapping Reductions

Definition: Let A and B be two languages. We say that there is a mapping reduction from A to B, and denote

$$A \leq_m B$$

if there is a computable function

$$f : \Sigma^* \longrightarrow \Sigma^*$$

such that, for every w,

Mapping Reductions

Definition: Let A and B be two languages. We say that there is a mapping reduction from A to B, and denote

$$A \leq_m B$$

if there is a computable function

$$f : \Sigma^* \longrightarrow \Sigma^*$$

such that, for every w,

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

The function f is called the reduction from A to B.

Mapping Reductions

Theorem:

If $A \leq_m B$ and B is decidable, then A is decidable.

Mapping Reductions

Theorem:

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: Let

- M be the decider for B, and
- f the reduction from A to B.

Mapping Reductions

Theorem:

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: Let

- M be the decider for B, and
- f the reduction from A to B.

Define N: On input w

1. compute $f(w)$
2. run M on input $f(w)$ and output whatever M outputs.

Mapping Reductions

Theorem:

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: Let

- M be the decider for B, and
- f the reduction from A to B.

Define N: On input w

1. compute $f(w)$
2. run M on input $f(w)$ and output whatever M outputs.

Corollary: If $A \leq_m B$ and A is undecidable, then B is undecidable.

Example: Primes

$$PRIMES = \{p | p \text{ is a prime}\}$$

P is a TM that accepts PRIME

$$INTEGERS = \{i | i \text{ is an integer}\}$$

$$\overline{A_{TM}} = \{\langle M, w\rangle | \text{TM } M \text{ does not accepts input } w\}$$

$$Prime_{TM} = \{\langle M\rangle | \text{M is a TM and } L(M) = PRIMES\}$$

Example: Primes

Define a computable function, f:

- input of form $\langle M, w \rangle$

Example: Primes

Define a computable function, f:

- input of form $\langle M, w \rangle$
- output of form $\langle M' \rangle$

Example: Primes

Define a computable function, f:

- input of form $\langle M, w \rangle$
- output of form $\langle M' \rangle$
- where $\langle M, w \rangle \in \overline{A_{TM}} \iff \langle M' \rangle \in Prime_{TM}$.

Example: Primes

The following machine computes this function f.
$F =$ on input $\langle M, w \rangle$:

- Construct the following machine M'.
  M': on input x
  - run P on x.
  - If P accepts x, $accept$.
  - if P rejects, run M on w.
  - If M accepts w $accept$.
- Output $\langle M' \rangle$

Example: Primes

- F is computable
- F does a correct mapping reduction:
- If $\langle M, w \rangle \in \overline{A_{TM}}$ then $L(M') = PRIMES$
- If $\langle M, w \rangle \notin \overline{A_{TM}}$ then $L(M') = INTEGERS$
- $Prime_{TM}$ is not in R.E. (since $\overline{A_{TM}}$ is not in R.E.)

Non Trivial Properties of $\mathcal{RE}$ Languages

A few examples

- L is finite.
- L is infinite.
- L contains the empty string.
- L contains no prime number.
- L is co-finite.
- . . .

All these are non-trivial properties of enumerable languages, since for each of them there is $L_1, L_2 \in \mathcal{RE}$ such that L_1 satisfies the property but L_2 does not.

Are there any trivial properties of $\mathcal{RE}$ languages?

Rice's Theorem

Theorem: Suppose $\mathcal{C}$ is a proper, non-empty subset of the set of enumerable languages, $\mathcal{RE}$, then it is undecidable whether for a given encoding of a TM, $\langle M \rangle$, $L(M)$ is in $\mathcal{C}$.

(See problem 5.22 in Sipser's book)

Rice's Theorem (Restated)

Theorem: Let $\mathcal{C}$ be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in $\mathcal{C}$. Then $L_{\mathcal{C}}$ is undecidable.

Proof: by reduction from A_{TM}.

Given M and w, we will construct M_0 such that:

- If M accepts w, then $\langle M_0 \rangle \in L_{\mathcal{C}}$.
- If M does not accept w, then $\langle M_0 \rangle \notin L_{\mathcal{C}}$.

Proof of Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- (Otherwise, look at $\overline{\mathcal{C}} = \mathcal{RE} \setminus \mathcal{C}$, also proper and non-empty.)
- Since $\mathcal{C}$ is not empty, there exists some language $L \in \mathcal{C}$. Let M_L be a TM accepting this language (recall $\mathcal{C} \subset \mathcal{RE}$ contains only recuresively enumerable languages).
- continued . . .

Proof of Rice's Theorem (cont.)

Given $\langle M, w \rangle$, construct M_0 such that:

- If M accepts w, then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept w, then $L(M_0) = \emptyset \notin \mathcal{C}$.

M_0 on input y:

1. Run M on w.
2. If M accepts w, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

Claim: The transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a mapping reduction from A_{TM} to $L_{\mathcal{C}}$.

Proof of Rice's Theorem (cont.^{2})

Proof: M_0 on input y:

1. Run M on w.
2. If M accepts, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

- The machine M_0 is simply a concatanation of two known TMs – the universal machine, and M_L.
- Therefore the transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a computable function, defined for all strings in Σ^*.
- (hey – what do we actually do with strings not of the form $\langle M, w \rangle$?)

Rice's Proof (Concluded)

- If $\langle M, w \rangle \in A_{TM}$ then M_0 gets to step 2, and runs M_L on y.
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, w \rangle \notin A_{TM}$ then M_0 never gets to step 2.
- In this case, $L(M_0) = \emptyset$, so $L(M_0) \notin \mathcal{C}$.
- This establishes the fact that $\langle M, w \rangle \in A_{TM}$ iff $\langle M_0 \rangle \in L_{\mathcal{C}}$. So we have $A_{TM} \leq_m L_{\mathcal{C}}$, thus $L_{\mathcal{C}}$ is undecidable. ♣

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
  - "does $L(M)$ contain infinitely many primes"
  - "does $L(M)$ contain an arithmetic progression of length 15"
  - "is $L(M)$ empty"

Rice's Theorem (Reflections)

- Decidability of properties related to the encoding itself cannot be inferred from Rice.

Rice's Theorem (Reflections)

- Decidability of properties related to the encoding itself cannot be inferred from Rice.
- For example "does $\langle M \rangle$ has an even number of states" is decidable.

Rice's Theorem (Reflections)

- Decidability of properties related to the encoding itself cannot be inferred from Rice.
- For example "does $\langle M \rangle$ has an even number of states" is decidable.
- Properties like "does M reaches state q_6 on the empty input string" are undecidable, but this does not follow from Rice's theorem.

Rice's Theorem (Reflections)

- Decidability of properties related to the encoding itself cannot be inferred from Rice.
- For example "does $\langle M \rangle$ has an even number of states" is decidable.
- Properties like "does M reaches state q_6 on the empty input string" are undecidable, but this does not follow from Rice's theorem.
- Likewise, Rice does not say anything on membership in $\mathcal{RE}$ of problems like "is $L(M)$ finite".

Rice's Theorem (Reflections)

- Decidability of properties related to the encoding itself cannot be inferred from Rice.
- For example "does $\langle M \rangle$ has an even number of states" is decidable.
- Properties like "does M reaches state q_6 on the empty input string" are undecidable, but this does not follow from Rice's theorem.
- Likewise, Rice does not say anything on membership in $\mathcal{RE}$ of problems like "is $L(M)$ finite".
- **Rice's Theorem is a powerful tool, but use it with care!**

Intro. to Controlled Executions

$$CET \quad = \quad \{\langle M, w, k\rangle | M \text{ accepts } w \text{ within } k \text{ steps}\}$$

Intro. to Controlled Executions

$$CET \quad = \quad \{\langle M, w, k\rangle | M \text{ accepts } w \text{ within } k \text{ steps}\}$$

Theorem: The language CET is decidable.

Intro. to Controlled Executions

$$CET = \{\langle M, w, k\rangle | M \text{ accepts } w \text{ within } k \text{ steps}\}$$

Theorem: The language CET is decidable.

What about space:

$$CES = \{\langle M, w, k\rangle | M \text{ accepts } w \text{ using } k \text{ cells}\}$$

Reductions via Controlled Executions

Consider the language $L_{\text{infinite}} = \{\langle M \rangle \mid L(M) \text{ is infinite}\}$.
By Rice Theorem, this language is not in $\mathcal{R}$.
We want to show that $L_{\text{infinite}} \notin RE$.

Reductions via Controlled Executions

Consider the language $\mathsf{L}_{\mathsf{infinite}} = \{\langle M \rangle \mid L(M) \text{ is infinite}\}$.
By Rice Theorem, this language is not in $\mathcal{R}$.
We want to show that $\mathsf{L}_{\mathsf{infinite}} \notin RE$.

Idea: Reduction from $\overline{H_{\mathsf{TM}}}$.
So we are after a reduction $f : \langle M, w \rangle \mapsto \langle M_0 \rangle$ such that

- If M halts on w then $L(M_0)$ is finite.
- If M does not halts on w then $L(M_0)$ is infinite.

Reductions via Controlled Executions

Consider the language $\mathsf{L}_{\mathsf{infinite}} = \{\langle M \rangle \mid L(M) \text{ is infinite}\}$.
By Rice Theorem, this language is not in $\mathcal{R}$.
We want to show that $\mathsf{L}_{\mathsf{infinite}} \notin RE$.

Idea: Reduction from $\overline{H_{\mathsf{TM}}}$.
So we are after a reduction $f : \langle M, w \rangle \mapsto \langle M_0 \rangle$ such that

- If M halts on w then $L(M_0)$ is finite.
- If M does not halts on w then $L(M_0)$ is infinite.

This looks a bit tricky...

Reductions via Controlled Executions (2)

We are after a reduction $f : \langle M, w \rangle \mapsto \langle M_0 \rangle$ such that

- If M halts on w then $L(M_0)$ is finite.
- If M does not halts on w then $L(M_0)$ is infinite.

Given $\langle M, w \rangle$, construct the TM M_0 as following:

- M_0 on input y
- Runs the universal machine U on $\langle M, w \rangle$ for $|y|$ steps.
- If U did not halt in that many steps, M_0 accepts y.
- If U did halt in that many steps, M_0 rejects y.

$f(\langle M, w \rangle) = M_0$. Let us examine $L(M_0)$.
(Remark: M_0 halts on all inputs.)

Reductions via Controlled Executions (3)

$f(\langle M, w \rangle) = M_0$. Let us examine $L(M_0)$.

Reductions via Controlled Executions (3)

$f(\langle M, w \rangle) = M_0$. Let us examine $L(M_0)$.

- If M does not halt on w, then M_0 accepts all y, so $L(M_0) = \Sigma^*$, and thus $\langle M_0 \rangle \in \mathsf{L}_{\mathsf{infinite}}$.
- If M does halt on w after k simulation steps, then M_0 accepts only ys of length smaller than k, so $L(M_0)$ is finite, and thus $\langle M_0 \rangle \notin \mathsf{L}_{\mathsf{infinite}}$.

Reductions via Controlled Executions (3)

$f(\langle M, w \rangle) = M_0$. Let us examine $L(M_0)$.

- If M does not halt on w, then M_0 accepts all y, so $L(M_0) = \Sigma^*$, and thus $\langle M_0 \rangle \in \mathsf{L}_{\mathsf{infinite}}$.
- If M does halt on w after k simulation steps, then M_0 accepts only ys of length smaller than k, so $L(M_0)$ is finite, and thus $\langle M_0 \rangle \notin \mathsf{L}_{\mathsf{infinite}}$.

We have shown that $\overline{H_{\mathsf{TM}}} \leq_m \mathsf{L}_{\mathsf{infinite}}$.

Since $\overline{H_{\mathsf{TM}}} \notin \mathcal{RE}$, this implies $\mathsf{L}_{\mathsf{infinite}} \notin \mathcal{RE}$. ♠

$\mathcal{RE}$-Completeness

Question: Is there a language L that is hardest in the class $\mathcal{RE}$ of enumerable languages (languages accepted by some TM)?

$\mathcal{RE}$-Completeness

Question: Is there a language L that is hardest in the class $\mathcal{RE}$ of enumerable languages (languages accepted by some TM)?

Answer: Well, you have to define what you mean by "hardest language".

$\mathcal{RE}$-Completeness

Question: Is there a language L that is hardest in the class $\mathcal{RE}$ of enumerable languages (languages accepted by some TM)?

Answer: Well, you have to define what you mean by "hardest language".

Definition: A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

$\mathcal{RE}$-Completeness

Definition A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

$\mathcal{RE}$-Completeness

Definition A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

The second item means that for every enumerable L there is a mapping reduction f_L from L to L_0. The reduction f_L depends on L and will typically differ from one language to another.

$\mathcal{RE}$-Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

$\mathcal{RE}$-Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

Theorem The language A_{TM} is $\mathcal{RE}$-Complete.

$\mathcal{RE}$-Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

Theorem The language A_{TM} is $\mathcal{RE}$-Complete.

Proof:

- The universal machine U accepts the language A_{TM}, so $A_{TM} \in \mathcal{RE}$.
- Suppose L is in $\mathcal{RE}$, and let M_L be a TM accepting it. Then $f_L(w) = \langle M_L, w \rangle$ is a mapping reduction from L to A_{TM} (why?). ♣

Reduction via Computation Histories

Important technique for proving undecidability.

- Useful for testing existence of some objects.

Reduction via Computation Histories

Important technique for proving undecidability.

- Useful for testing existence of some objects.
- For example, basis for proof of undecidability in Hilbert's tenth problem,

Reduction via Computation Histories

Important technique for proving undecidability.

- Useful for testing existence of some objects.
- For example, basis for proof of undecidability in Hilbert's tenth problem,
- where "object" is integral root of polynomial.

Reduction via Computation Histories

Important technique for proving undecidability.

- Useful for testing existence of some objects.
- For example, basis for proof of undecidability in Hilbert's tenth problem,
- where "object" is integral root of polynomial.
- Other examples: Does a context free grammar generate Σ^*?

Reduction via Computation Histories

Important technique for proving undecidability.

- Useful for testing existence of some objects.
- For example, basis for proof of undecidability in Hilbert's tenth problem,
- where "object" is integral root of polynomial.
- Other examples: Does a context free grammar generate Σ^*?
- Does a linear bounded TM accept the empty language?

Reminder: Configurations

Configuration:

$$1011q_70111$$

means:

- state is q_7
- LHS of tape is 1011
- RHS of tape is 0111
- head is on RHS 0

Configurations

- configuration uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$
- Of course, uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$
- Special case (left end of tape): q_ibv yields q_jcv if $\delta(q_i, b) = (q_j, c, L)$.

Computation Histories

Let M be a TM and w an input string.

- An accepting computation history for M on w is a sequence $C_1, C_2, \ldots, C_\ell$, where
  - C_1 is the starting configuration of M on w
  - C_ℓ is an accepting configuration of M,
  - each C_i yields C_{i+1} according to the transition function.

Computation Histories

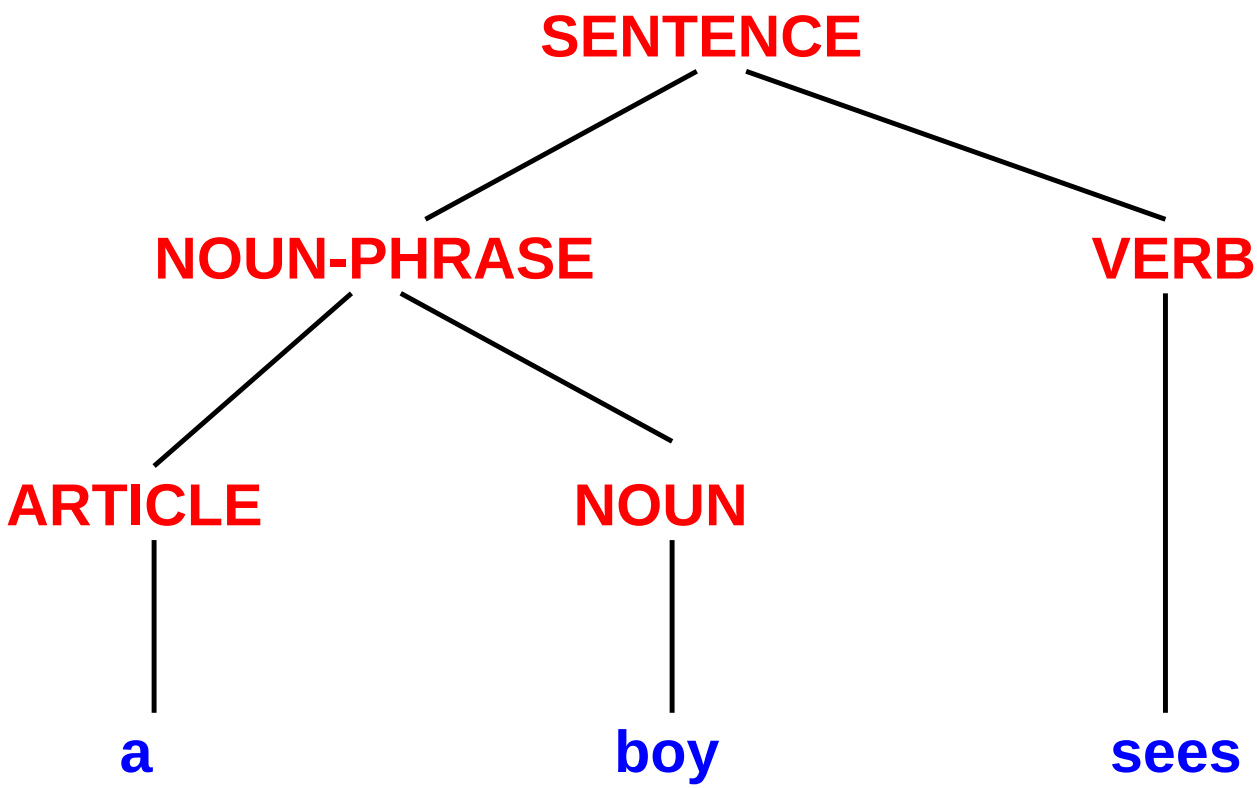
Let M be a TM and w an input string.

- An accepting computation history for M on w is a sequence $C_1, C_2, \ldots, C_\ell$, where
  - C_1 is the starting configuration of M on w
  - C_ℓ is an accepting configuration of M,
  - each C_i yields C_{i+1} according to the transition function.
- A rejecting computation history for M on w is the same, except
  - C_ℓ is a rejecting configuration of M.

Remarks

- Computation sequences are finite.
- If M does not halt on w, no accepting or rejecting computation history exists.
- Notion is useful for both deterministic (one history) and non-deterministic (many histories) TMs.

A CFG Question



Emptiness of CFGs

We have already seen an algorithm to check whether a context-free grammar is empty.

On input $\langle G \rangle$ where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables become marked:
3. Mark any A where

$$A \to U_1 U_2 \ldots U_k$$

   and each U_i has already been marked.

4. If start symbol marked, accept, otherwise reject.

Using Computation Histories for CFGs

So the language $\text{EMPTY}_{\text{CFG}}$ is decidable.

Question: What about the complementary question: Does a CFG generate all strings?

Using Computation Histories for CFGs

So the language $\text{EMPTY}_{\text{CFG}}$ is decidable.

Question: What about the complementary question: Does a CFG generate all strings?

$$\text{All}_{\text{CFG}} = \{\langle G \rangle | G \text{ is a CFL and } L(G) = \Sigma^*\}$$

Does a CFG Generate All Strings?

Theorem: All_{CFG} is undecidable.

Proof by reduction from A_{TM} to $\overline{\text{All}_{\text{CFG}}}$:

- Given $\langle M, w \rangle$, construct a coding of a CFG, $\langle G \rangle$
- G generates all strings that are not accepting computation histories for M on w
- if M does not accept w, G generates all strings
- if M does accept w, G does not generate the accepting computation history.

Does a CFG Generate All Strings?

An accepting computation history appears as $\#C_1\#C_2\#\ldots\#C_\ell\#$, where

- C_1 is the starting configuration of M on w,
- C_ℓ is an accepting configuration of M,
- Each C_i yields C_{i+1} by transition function of M.

A string is not an accepting computation history if it fails one or more of these conditions.

Does a CFG Generate All Strings?

Instead of the CFG, G, we construct a PDA, D (recall equivalence).
D non-deterministically "guesses" which condition is violated.

- then verifies the guessed violation:
  - Is there some C_i that is not a configuration of M (number of q symbols $\neq 1$)?
  - Is C_1 not the starting configuration of M on w?
  - Is C_ℓ not an accepting configuration of M?
  - Does C_i not yield C_{i+1} by the transition function of M?
- The last condition is the tricky one to check.

Does a CFG Generate All Strings?

- Does C_i not yield C_{i+1}?

Idea:

- Scan input. Nondeterministically decide "violating configuration" C_i was reached.
- Push C_i onto the stack till $\#$.
- scan C_{i+1} and pop matching symbols of C_i
  - check if C_i and C_{i+1} match everywhere, except . . .
  - around the head position,
  - where difference dictated by transition function for M.

Houston, We Have a Problem

When D pops C_i from stack, C_i is in reverse order. Ignoring the local changes around head position, what we were trying to identify the language $x\#y$, with $x \neq y$.

While this can be done in principle by a non deterministic PDA (see problem 2.26 in Sipser's book), there is a simpler way.

Houston, We Have a Problem

When D pops C_i from stack, C_i is in reverse order. Ignoring the local changes around head position, what we were trying to identify the language $x\#y$, with $x \neq y$.

While this can be done in principle by a non deterministic PDA (see problem 2.26 in Sipser's book), there is a simpler way.

Houston, We Have a Problem

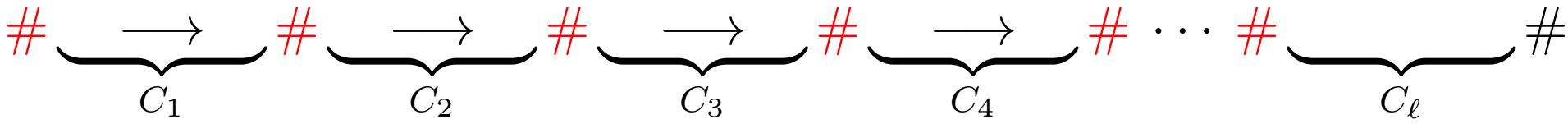
When D pops C_i from stack, C_i is in reverse order. Ignoring the local changes around head position, what we were trying to identify the language $x\#y$, with $x \neq y$.

While this can be done in principle by a non deterministic PDA (see problem 2.26 in Sipser's book), there is a simpler way.

So far, we used a "straight" notion of accepting computation histories

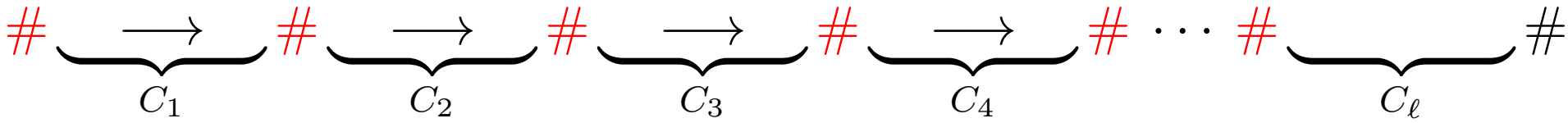
Does a CFG Generate All Strings?

So far, we used a "straight" notion of accepting computation histories



Does a CFG Generate All Strings?

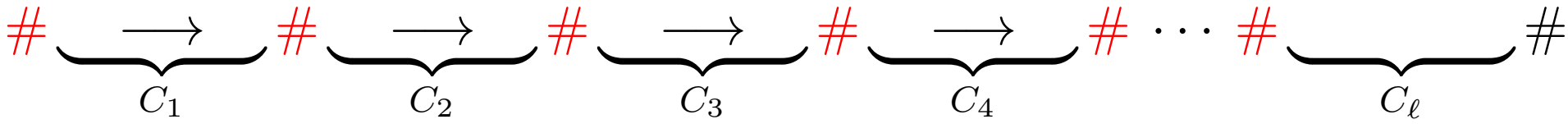
So far, we used a "straight" notion of accepting computation histories



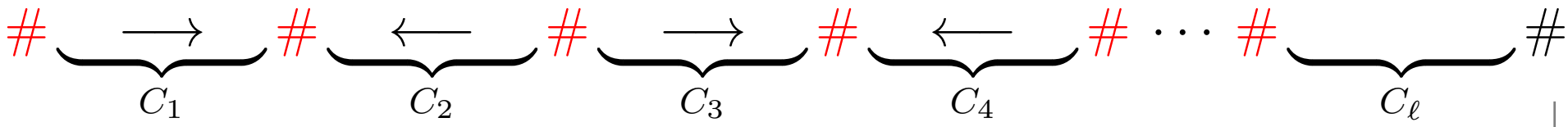
But in this modern age, why not employ an alternative notion of accepting computation history, one that will make the life of our PDA much easier?

Does a CFG Generate All Strings?

So far, we used a "straight" notion of accepting computation histories



But in this modern age, why not employ an alternative notion of accepting computation history, one that will make the life of our PDA much easier? **Solution:** Write the accepting computation history so that every other configuation is in reverse order.



This takes care of difficulty in the proof.

Wrapping Things Up

Given $\langle M, w \rangle$, we constructed (algorithmically) a PDA, D, which rejects the string z if and only if z equals an accepting computation history of M on w, written in the "alternating format".

Wrapping Things Up

Given $\langle M, w \rangle$, we constructed (algorithmically) a PDA, D, which rejects the string z if and only if z equals an accepting computation history of M on w, written in the "alternating format".

Therefore $L(D)$ is either Σ^* or $\Sigma^* \setminus \{z\}$.

Wrapping Things Up

Given $\langle M, w \rangle$, we constructed (algorithmically) a PDA, D, which rejects the string z if and only if z equals an accepting computation history of M on w, written in the "alternating format".

Therefore $L(D)$ is either Σ^* or $\Sigma^* \setminus \{z\}$.

This D has an equivalent (and efficiently described) CFG, G, namely $L(D) = L(G)$. So $L(G)$ is either Σ^* or $\Sigma^* \setminus \{z\}$. The mapping $\langle M, w \rangle \mapsto \langle G \rangle$ is thus a reduction from A_{TM} to $\overline{\text{All}_{\text{CFG}}}$.

Wrapping Things Up

Given $\langle M, w \rangle$, we constructed (algorithmically) a PDA, D, which rejects the string z if and only if z equals an accepting computation history of M on w, written in the "alternating format".

Therefore $L(D)$ is either Σ^* or $\Sigma^* \setminus \{z\}$.

This D has an equivalent (and efficiently described) CFG, G, namely $L(D) = L(G)$. So $L(G)$ is either Σ^* or $\Sigma^* \setminus \{z\}$. The mapping $\langle M, w \rangle \mapsto \langle G \rangle$ is thus a reduction from A_{TM} to $\overline{\mathsf{All}_{\mathsf{CFG}}}$.

Since $A_{TM} \notin \mathcal{R}$ we get $\overline{\mathsf{All}_{\mathsf{CFG}}} \notin \mathcal{R}$.

Wrapping Things Up

Given $\langle M, w \rangle$, we constructed (algorithmically) a PDA, D, which rejects the string z if and only if z equals an accepting computation history of M on w, written in the "alternating format".

Therefore $L(D)$ is either Σ^* or $\Sigma^* \setminus \{z\}$.

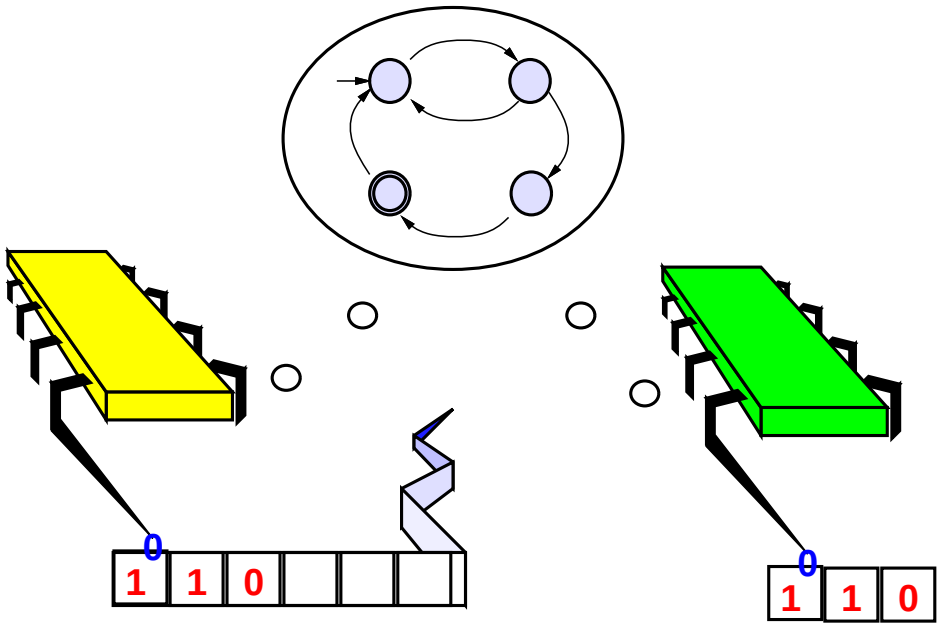
This D has an equivalent (and efficiently described) CFG, G, namely $L(D) = L(G)$. So $L(G)$ is either Σ^* or $\Sigma^* \setminus \{z\}$. The mapping $\langle M, w \rangle \mapsto \langle G \rangle$ is thus a reduction from A_{TM} to $\overline{\mathsf{All}_{\mathsf{CFG}}}$.

Since $A_{TM} \notin \mathcal{R}$ we get $\overline{\mathsf{All}_{\mathsf{CFG}}} \notin \mathcal{R}$.

As the class $\mathcal{R}$ is closed under complement, we conclude that $\mathsf{All}_{\mathsf{CFG}} \notin \mathcal{R}$.

Linear Bounded Automata

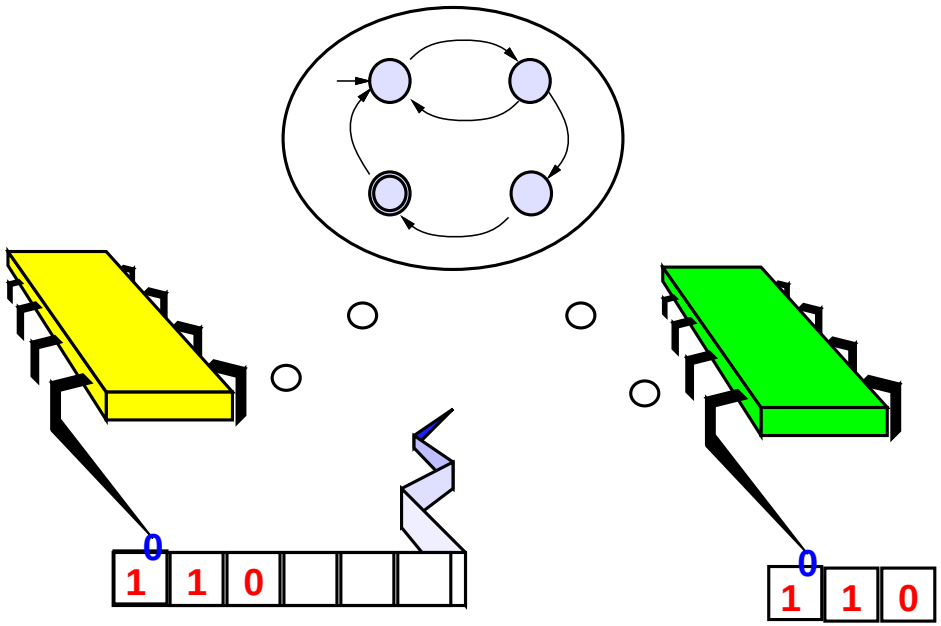
- A restricted form of TM.



turing machine **linear bounded automaton**

Linear Bounded Automata

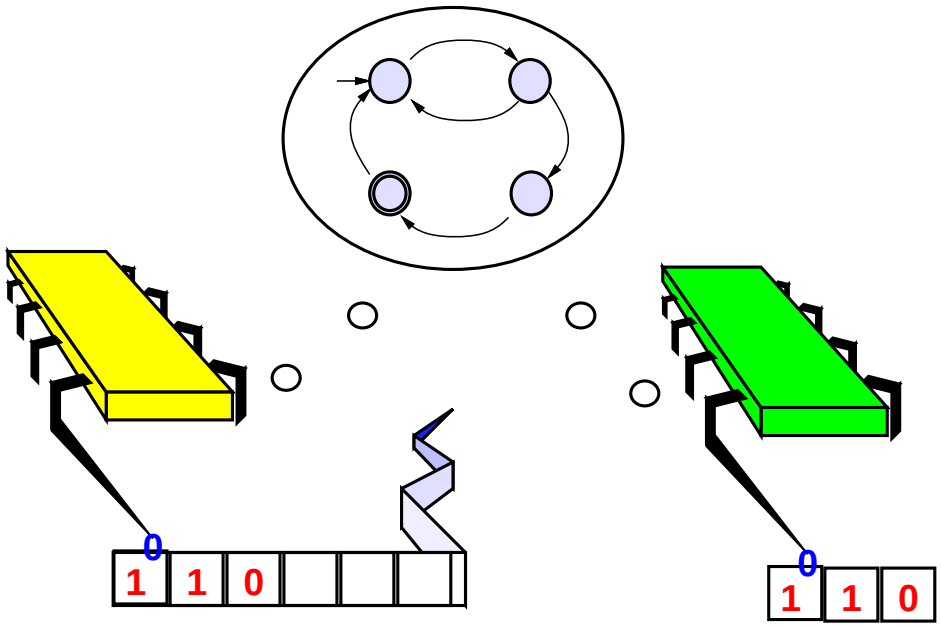
- A restricted form of TM.
- Cannot move off portion of tape containing input



turing machine **linear bounded automaton**

Linear Bounded Automata

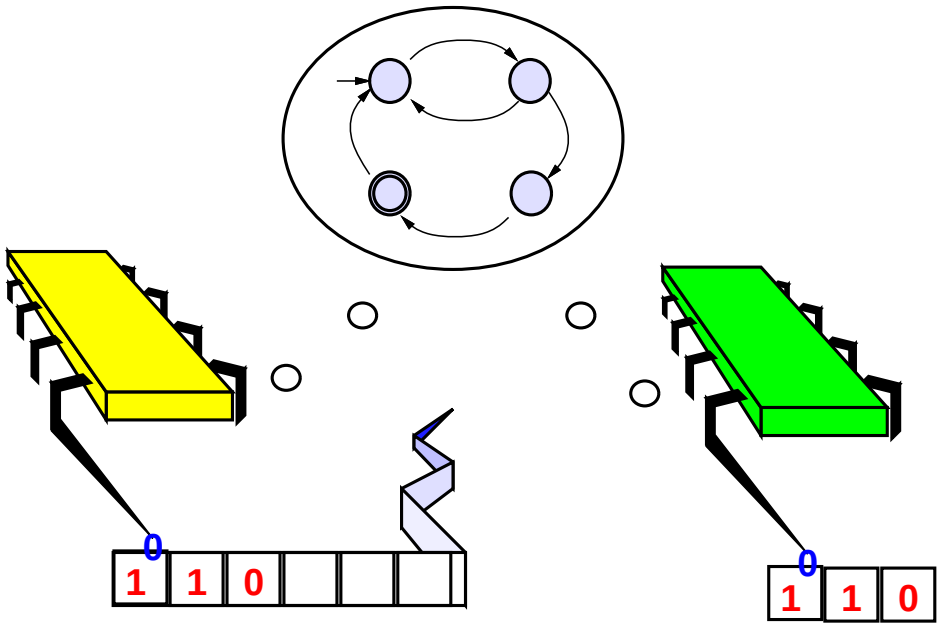
- A restricted form of TM.
- Cannot move off portion of tape containing input
- Rejects attempts to move head beyond input



turing machine **linear bounded automaton**

Linear Bounded Automata

- A restricted form of TM.
- Cannot move off portion of tape containing input
- Rejects attempts to move head beyond input
- Size of input determines size of memory



turing machine **linear bounded automaton**

Linear Bounded Automata

Question: Why linear?

Answer: Using a tape alphabet larger than the input alphabet increases memory by a constant factor.

Linear Bounded Automata

Believe it or not, LBAs are quite powerful.

The deciders for

- A_{DFA} (does a DFA accept a string?)
- A_{CFG} (is string in a CFG?)
- $\text{EMPTY}_{\text{DFA}}$ (is a DFA trivial?)
- $\text{EMPTY}_{\text{CFG}}$ (is a CFL empty?)

are all LBAs.

Every CFL can be decided by a LBA.

Not too easy to find a natural, decidable language that cannot be decided by an LBA.

Acceptance for LBAs

Define

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

Question: Is A_{LBA} decidable?

Answer: Unlike A_{TM}, the language A_{LBA} is decidable!

Lemma:

Let M be a LBA with

- q states
- g symbols in tape alphabet

On an input of size n, LBA has exactly qng^n distinct configurations, because a configuration involves:

Lemma:

Let M be a LBA with

- q states
- g symbols in tape alphabet

On an input of size n, LBA has exactly qng^n distinct configurations, because a configuration involves:

- control state (q possibilities)

Lemma:

Let M be a LBA with

- q states
- g symbols in tape alphabet

On an input of size n, LBA has exactly qng^n distinct configurations, because a configuration involves:

- control state (q possibilities)
- head position (n possibilities)

Lemma:

Let M be a LBA with

- q states
- g symbols in tape alphabet

On an input of size n, LBA has exactly qng^n distinct configurations, because a configuration involves:

- control state (q possibilities)
- head position (n possibilities)
- tape contents (g^n possibilities)

Theorem: A_{LBA} is decidable

Idea:

- Simulate M on w (if M tries to "exit" the input space, halt and reject).
- But what do we do if M loops?
- Must detect looping and reject.
- M loops if and only if it repeats a configuration.
- Why? And is this also true of "regular" TMs?
- By pigeon hole, if our LBA M runs long enough, it must repeat a configuration!

Theorem: A_{LBA} is decidable

On input $\langle M, w \rangle$, where M is an LBA and $w \in \Sigma^*$,

1. Simulate M on w,
2. while maintaining a counter.
3. Counter incremented by 1 per each simulated step (of M).
4. Keep simulating M for qng^n steps, or until it halts (whichever comes first)
5. If M has halted, accept w if it was accepted by M, and reject w if it was rejected by M.
6. reject w if counter limit reached (M has not halted).

More LBAs

Surprisingly though, LBAs do have undecidable problems too!

Here is a related problem.

$$\text{Non-EMPTY}_{\text{LBA}} = \{\langle M \rangle | M \text{ is an LBA and } L(M) \neq \emptyset\}$$

Question: Is $\text{Non-EMPTY}_{\text{LBA}}$ decidable?

Non-EMPTY$_{\text{LBA}}$

$$\text{Non-EMPTY}_{\text{LBA}} = \{\langle M \rangle | M \text{ is an LBA and } L(M) \neq \emptyset\}$$

Theorem: Non-EMPTY$_{\text{LBA}}$ is undecidable.

Proof by reduction from A_{TM}, using computation histories.

More LBAs

Given M and w, we will construct an LBA, B.

- $L(B)$ will contain exactly all accepting computation histories for M on w.
- M accepts w iff $L(B) \neq \emptyset$.

More LBAs

It is not enough to show that B exists.

We must show that the mapping from $\langle M, w \rangle$ to $\langle B \rangle$ is computable.

We are now going to describe the linear bounded machine, $\langle B \rangle$. It will be clear that indeed $\langle B \rangle$ is computable from $\langle M, w \rangle$.

Assume an accepting computation history is presented as a string:

$$\#\underbrace{\qquad}_{C_1}\#\underbrace{\qquad}_{C_2}\#\underbrace{\qquad}_{C_3}\#\cdots\#\underbrace{\qquad}_{C_\ell}\#,$$

with descriptions of configurations separated by $\#$ delimiters.

The LBA

The LBA, B, works as follows:

On input x, the LBA B:

- breaks x according to the $\#$ delimiters
- identifies strings $C_1, C_2, \ldots, C_\ell$.
- then checks that all the following conditions hold:
  - Each C_i are a configuration of M
  - C_1 is the start configuration of M on w
  - Every C_{i+1} follows from C_i according to M
  - C_ℓ is an accepting configuration

The LBA

- Checking that each C_i is a configuration of M is easy: All it means is that C_i includes exactly one q symbols.

The LBA

- Checking that each C_i is a configuration of M is easy: All it means is that C_i includes exactly one q symbols.
- Checking that C_1 is the start configuration on w, $q_0 w_1 w_2 \cdots w_n$, is easy, because the string w is "wired into" B.

The LBA

- Checking that each C_i is a configuration of M is easy: All it means is that C_i includes exactly one q symbols.
- Checking that C_1 is the start configuration on w, $q_0 w_1 w_2 \cdots w_n$, is easy, because the string w is "wired into" B.
- Checking that C_ℓ is an accepting configuration is easy, because C_ℓ must include the accepting state q_a.

The LBA

- Checking that each C_i is a configuration of M is easy: All it means is that C_i includes exactly one q symbols.
- Checking that C_1 is the start configuration on w, $q_0 w_1 w_2 \cdots w_n$, is easy, because the string w is "wired into" B.
- Checking that C_ℓ is an accepting configuration is easy, because C_ℓ must include the accepting state q_a.
- The only hard part is checking that each C_{i+1} follows from C_i by M's transition function.

The Hard Part

Checking that for all i, C_{i+1} follows from C_i by M's transition function:

- C_i and C_{i+1} almost identical, except for positions under head and adjacent to head.
- These positions should updated according to transition function.

Do this verification by

- zig-zagging between corresponding positions of C_i and C_{i+1}.
- use "dots" on tape to mark current position
- all this can be done inside space allocated by input x. Thus B is indeed a LBA.

Important!

The LBA, B, accepts the string x if and only if x equals an accepting computation history of M on w.

Therefore $L(B)$ is either empty or a singleton $\{x\}$.

We construct B so that $L(B)$ is non-empty iff M accepts w.

Thus $\langle M, w\rangle \in A_{TM}$ iff $\langle B\rangle \in$ Non-EMPTY$_{\text{LBA}}$.

Namely $A_{TM} \leq_m A_{\text{LBA}}$, so Non-EMPTY$_{\text{LBA}} \notin \mathcal{R}$. ♠

BTW, is Non-EMPTY$_{\text{LBA}} \in \mathcal{RE}$?