Nondeterminism adds power to PDA (revised)
Computational Models - Lecture 5

- Nondeterminism adds power to PDA (revised)
- Closure Properties of CFLs
Computational Models - Lecture 5

- Nondeterminism adds power to PDA (revised)
- Closure Properties of CFLs
- Algorithmic Aspects of PDAs and CFLs
Computational Models - Lecture 5

- Nondeterminism adds power to PDA (revised)
- Closure Properties of CFLs
- Algorithmic Aspects of PDAs and CFLs
- DFAs and PDAs: Perspectives
Computational Models - Lecture 5

- Nondeterminism adds power to PDA (revised)
- Closure Properties of CFLs
- Algorithmic Aspects of PDAs and CFLs
- DFAs and PDAs: Perspectives
- Turing Machines
Computational Models - Lecture 5

- Nondeterminism *adds power* to PDA (revised)
- Closure Properties of CFLs
- Algorithmic Aspects of PDAs and CFLs
- DFAs and PDAs: Perspectives
- *Turing* Machines
Computational Models - Lecture 5

- Nondeterminism adds power to PDA (revised)
- Closure Properties of CFLs
- Algorithmic Aspects of PDAs and CFLs
- DFAs and PDAs: Perspectives
- Turing Machines

- Sipser’s book, 2.2, 2.3, & 3.1 (not all material from book)
Mid-term exam on Friday, Nov. 26
Mid-term exam on Friday, Nov. 26
No more date changes will occur (this semester)
Mid-term exam on Friday, Nov. 26

- No more date changes will occur (this semester)
- Material includes first five lectures (i.e. up to and including today)
Mid-term exam on Friday, Nov. 26

- No more date changes will occur (this semester)
- Material includes first five lectures (i.e. up to and including today)
- Can use 4 pages, double sided, any size font you can read without external magnification devices.
NonDeterminism – Corrected Proof

Theorem: Let $M$ be a PDA that accepts

$$L = \{x^n y^n | n \geq 0\} \cup \{x^n y^{2n} | n \geq 0\}.$$ 

Then $M$ is non-deterministic.

\[a\] (prf modified from [www.cs.may.ie/~jpower/Courses/parsing/node38.html])
NonDeterminism – Corrected Proof

Theorem: Let $M$ be a PDA that accepts

$$L = \{x^n y^n | n \geq 0\} \cup \{x^n y^{2n} | n \geq 0\}.$$ 

Then $M$ is non-deterministic.

Proof: Suppose, by way of contradiction, that $M$ is deterministic.

(prf modified from www.cs.may.ie/~jpower/Courses/parsing/node38.html)
NonDeterminism – Corrected Proof

Theorem: Let $M$ be a PDA that accepts

$$L = \{x^n y^n \mid n \geq 0\} \cup \{x^n y^{2n} \mid n \geq 0\}.$$

Then $M$ is non-deterministic.

Proof: Suppose, by way of contradiction, that $M$ is deterministic.

- Create two copies of this PDA, denoted $M_1$ and $M_2$.
- Two states in $M_1$ and $M_2$ are called “cousins” if they are copies of the same state in the original PDA.

(prf modified from [www.cs.may.ie/~jpower/Courses/parsing/node38.html](http://www.cs.may.ie/~jpower/Courses/parsing/node38.html))
NonDeterminism Essential (cont.)

We now modify these copies to make them into one PDA, $M_0$, over the alphabet \( \{x, y, z\} \).
NonDeterminism Essential (cont.)

- We now modify these copies to make them into one PDA, $M_0$, over the alphabet $\{x, y, z\}$.
- States of the new $M_0$ are those of $M_1$ union $M_2$. 
NonDeterministm Essential (cont.)

- We now modify these copies to make them into one PDA, \( M_0 \), over the alphabet \( \{x, y, z\} \).
- States of the new \( M_0 \) are those of \( M_1 \) union \( M_2 \).
- Start state of the new \( M_0 \) is the start state of \( M_1 \).
NonDeterminism Essential (cont.)

- We now modify these copies to make them into one PDA, $M_0$, over the alphabet \{x, y, z\}.
- States of the new $M_0$ are those of $M_1$ union $M_2$.
- Start state of the new $M_0$ is the start state of $M_1$.
- The accepting states of the new $M_0$ are the accepting states of $M_2$. 
NonDeterminism Essential (cont.)

Modifications:
NonDeterminism Essential (cont.)

- Modifications:
  - Erase all $x$ transitions of $M_2$. 
NonDeterminism Essential (cont.)

- Modifications:
  - Erase all $x$ transitions of $M_2$.
  - Replace every existing $y$ transition of $M_2$ by a new $z$ transition.
NonDeterminism Essential (cont.)

- Modifications:
  - Erase all \( x \) transitions of \( M_2 \).
  - Replace every existing \( y \) transition of \( M_2 \) by a new \( z \) transition.
  - At this point \( M_2 \) got only \( z \) transition (so \( x \) and \( y \) inputs lead immediately to rejection).
NonDeterminism Essential (cont.)

- Modifications:
  - Erase all $x$ transitions of $M_2$.
  - Replace every existing $y$ transition of $M_2$ by a new $z$ transition.
  - At this point $M_2$ got only $z$ transition (so $x$ and $y$ inputs lead immediately to rejection).
  - Erase all $x$ transitions out of accept states of $M_1$. 
The surgery is almost done, but if we don’t connect the two halves of its brain, the patient will not function coherently.
NonDeterminism Essential (cont.)

- The surgery is almost done, but if we don’t connect the two halves of its brain, the patient will not function coherently.

- Replace every existing $y$ transition leading out of accept states of $M_1$ by a new $z$ transition, and redirect it to its “cousin” in $M_2$. 
NonDeterminism Essential (cont.)

- The surgery is almost done, but if we don’t connect the two halves of its brain, the patient will not function coherently.

- Replace every existing $y$ transition leading out of accept states of $M_1$ by a new $z$ transition, and redirect it to its “cousin” in $M_2$. 
NonDeterminism Essential (cont.)

- The surgery is almost done, but if we don’t connect the two halves of its brain, the patient will not function coherently.

- Replace every existing $y$ transition leading out of accept states of $M_1$ by a new $z$ transition, and redirect it to its “cousin” in $M_2$.

- Surgery over. Patient (a deterministic PDA) still alive. Let us now diagnose what, if anything, it can do.
NonDeterminism Essential (cont.)

- What language $M_0$ recognizes?
NonDeterminism Essential (cont.)

- What language $M_0$ recognizes?
- Certainly if $M_0$ accepts a string, it must be of the form $(x \cup y)^* z^*$. 
What language $M_0$ recognizes?

Certainly if $M_0$ accepts a string, it must be of the form $(x \cup y)^* z^*.$

But surely not all strings of that form are accepted by $M_0.$
NonDeterminism Essential (cont.)

- What language $M_0$ recognizes?

- Certainly if $M_0$ accepts a string, it must be of the form $(x \cup y)^* z^*$.

- But surely not all strings of that form are accepted by $M_0$.

- For example, the $(x \cup y)^*$ prefix must be accepted by the original $M$. 
NonDeterminism Essential (cont.)

- What language $M_0$ recognizes?
- Certainly if $M_0$ accepts a string, it must be of the form $(x \cup y)^* z^*$. 
- But surely not all strings of that form are accepted by $M_0$. 
- For example, the $(x \cup y)^*$ prefix must be accepted by the original $M$. 
- Otherwise there will be no switch to $M_2$, and no acceptance by $M_0$. (why? would this also be true for non deterministic $M$?)
NonDeterminism Essential (cont.)

- What language $M_0$ recognizes?
- Certainly if $M_0$ accepts a string, it must be of the form $(x \cup y)^*z^*$.
- But surely not all strings of that form are accepted by $M_0$.
- For example, the $(x \cup y)^*$ prefix must be accepted by the original $M$.
- Otherwise there will be no switch to $M_2$, and no acceptance by $M_0$. (why? would this also be true for non deterministic $M$?)
- So the prefix of an accepted string is either of the form $x^n y^n$ or $x^n y^{2n}$. 
NonDeterminism Essential (cont.)

- What can we say about the $z^i$ part? First, $i$ must be greater than 0 for a transition to take place.
NonDeterminism Essential (cont.)

- What can we say about the $z^i$ part? First, $i$ must be greater than 0 for a transition to take place.
- By construction, $M_2$ on $z^i$ imitates the actions of $M$ on $y^i$ from the same starting point.
NonDeterminism Essential (cont.)

- What can we say about the $z^i$ part? First, $i$ must be greater than 0 for a transition to take place.

- By construction, $M_2$ on $z^i$ imitates the actions of $M$ on $y^i$ from the same starting point.

- This means that if $M_0$ accepts $x^n y^n z^i$ then $M$ accepts $x^n y^{n+i}$.
NonDeterminism Essential (cont.)

- What can we say about the $z^i$ part? First, $i$ must be greater than 0 for a transition to take place.

- By construction, $M_2$ on $z^i$ imitates the actions of $M$ on $y^i$ from the same starting point.

- This means that if $M_0$ accepts $x^ny^n z^i$ then $M$ accepts $x^ny^{n+i}$.

- Which is possible if either $i = n$, so $M_0$ accepts $x^ny^n z^n$, $n > 0$. 
NonDeterminism Essential (cont.)

- What can we say about the \( z^i \) part? First, \( i \) must be greater than 0 for a transition to take place.

- By construction, \( M_2 \) on \( z^i \) imitates the actions of \( M \) on \( y^i \) from the same starting point.

- This means that if \( M_0 \) accepts \( x^n y^n z^i \) then \( M \) accepts \( x^n y^{n+i} \).

- Which is possible if either \( i = n \), so \( M_0 \) accepts \( x^n y^n z^n \), \( n > 0 \).

- Or \( M_0 \) accepts \( x^n y^{2n} z^j \), so \( M \) accepts \( x^n y^{2n+j} \).
NonDeterminism Essential (cont.)

- What can we say about the $z^i$ part? First, $i$ must be greater than 0 for a transition to take place.

- By construction, $M_2$ on $z^i$ imitates the actions of $M$ on $y^i$ from the same starting point.

- This means that if $M_0$ accepts $x^ny^n z^i$ then $M$ accepts $x^ny^{n+i}$.

- Which is possible if either $i = n$, so $M_0$ accepts $x^ny^n z^n$, $n > 0$.

- Or $M_0$ accepts $x^ny^{2n} z^j$, so $M$ accepts $x^ny^{2n+j}$.

- But $L$ contains no strings of this last form!
Conclusion of Proof

- We just showed that the PDA $M_0$ accepts the language $\{x^n y^n z^n | n \geq 1\}$. Contradiction.
Conclusion of Proof

- We just showed that the PDA $M_0$ accepts the language $\{x^n y^n z^n | n \geq 1\}$. Contradiction.

- Contradiction? What contradiction? What the $\%\&\#$ does this contradict?
Conclusion of Proof

- We just showed that the PDA $M_0$ accepts the language $\{x^n y^n z^n \mid n \geq 1\}$. Contradiction. ♣

- Contradiction? What contradiction? What the $\%\&\#$ does this contradict?

- It contradicts the fact that by the so called $uvwxyz$ pumping lemma, the language $\{x^n y^n z^n \mid n \geq 1\}$ is not context free, so is not accepted by a PDA.
Conclusion of Proof

- We just showed that the PDA $M_0$ accepts the language $\{x^ny^n z^n | n \geq 1\}$. Contradiction.

- Contradiction? What contradiction? What the $%&#$ does this contradict?

- It contradicts the fact that by the so called $uvwxyz$ pumping lemma, the language $\{x^ny^nz^n | n \geq 1\}$ is not context free, so is not accepted by a PDA.

- So our initial supposition that the language $\{x^ny^n\} \cup \{x^ny^{2n}\}$ is accepted by a deterministic PDA, does not hold.
Conclusion of Proof

- We just showed that the PDA $M_0$ accepts the language $\{x^n y^n z^n | n \geq 1\}$. Contradiction.

- Contradiction? What contradiction? What the %& does this contradict?

- It contradics the fact that by the so called $uvwxyz$ pumping lemma, the language $\{x^n y^n z^n | n \geq 1\}$ is not context free, so is not accepted by a PDA.

- So our initial supposition that the language $\{x^n y^n\} \cup \{x^n y^{2n}\}$ is accepted by a deterministic PDA, does not hold.

- While thinking about the proof, where would it fail if the original $M$ were non-deterministic?
A Couple of Reminders
Equivalence Theorem (reminder)

**Theorem:** A language is context free if and only if some pushdown automaton accepts it.

This time, both the “if” part and the “only if” part are non-trivial.
Pumping Lemma for CFL

Also known as the $uvxyz$ Theorem.

**Theorem:** If $A$ is a CFL, there is an $\ell$ (critical length), such that if $s \in A$ and $|s| \geq \ell$, then $s = uvxyz$ where

- for every $i \geq 0$, $uv^i xy^i z \in A$
Pumping Lemma for CFL

Also known as the $uvxyz$ Theorem.

**Theorem:** If $A$ is a CFL, there is an $\ell$ (critical length), such that if $s \in A$ and $|s| \geq \ell$, then $s = uvxyz$ where

- for every $i \geq 0$, $uv^i xy^i z \in A$
- $|vy| > 0$, (non-triviality)
Pumping Lemma for CFL

Also known as the uvxyz Theorem.

**Theorem:** If $A$ is a CFL, there is an $\ell$ (critical length), such that if $s \in A$ and $|s| \geq \ell$, then $s = uvxyz$ where

- for every $i \geq 0$, $uv^ixyz \in A$
- $|vy| > 0$, (non-triviality)
- $|vxy| \leq \ell$. 
Proof (Sketch)

Split $s = uvxyz$
Proof (Sketch)

Split $s = uvxyz$

Each occurrence of $R$ produces a string
Proof (Sketch)

Split $s = uvxyz$

- each occurrence of $R$ produces a string
- upper produces string $vxy$
Proof (Sketch)

Split \( s = uvxyz \)

Each occurrence of \( R \) produces a string
- upper produces string \( vxy \)
- lower produces string \( x \)
Proof (Sketch 2)

Replacing smaller by larger yields $uv^ixy^iz$, for $i > 0$. 
Proof (Sketch 3)

Replacing larger by smaller yields $uxz$.

Together, they establish:

$$\text{for } i \geq 0, \ uv^i xy^i z \in A$$
Proof (Sketch 4)

Next condition is:

- $|vy| > 0$

If $v$ and $y$ are both $\varepsilon$, then

is a parse tree for $s$ with fewer nodes.
Another CF Pumping Lemma Example

**Theorem:** \( L = \{ a^p : p \text{ is prime} \} \) is not context free
CF Pumping Lemma

\[ L = \{ a^p : p \text{ is prime} \} \text{ is not context free} \]

Let \( \ell \) be the constant of the context free languages pumping lemma
CF Pumping Lemma

\[ L = \{ a^p : p \text{ is prime} \} \text{ is not context free} \]

- Let \( \ell \) be the constant of the context free languages pumping lemma
- Consider \( w = a^p \), where \( p \) is the smallest prime number \( > \ell \).
CF Pumping Lemma

\[ L = \{a^p : p \text{ is prime} \} \] is not context free

- Let \( \ell \) be the constant of the context free languages pumping lemma
- Consider \( w = a^p \), where \( p \) is the smallest prime number \( > \ell \).
- If we break \( w = uvxyz \):
CF Pumping Lemma

\[ L = \{ a^p : p \text{ is prime} \} \text{ is not context free} \]

- Let \( \ell \) be the constant of the context free languages pumping lemma.
- Consider \( w = a^p \), where \( p \) is the smallest prime number \( > \ell \).
- If we break \( w = uvxyz \):
  - Let \( |vy| = k, k > 1 \), and \( |uxz| = r = \ell - k \).
CF Pumping Lemma

\[ L = \{a^p : p \text{ is prime} \} \] is not context free

- Let \( \ell \) be the constant of the context free languages pumping lemma
- Consider \( w = a^p \), where \( p \) is the smallest prime number \( > \ell \).
- If we break \( w = uvxyz \):
  - Let \( |vy| = k, k > 1 \), and \( |uxz| = r = \ell - k \)
  - \( uv^i xy^i z = a^{r+ik} \), so \( \forall i, r + ik \) must be prime
CF Pumping Lemma

\[ L = \{a^p : p \text{ is prime} \} \text{ is not context free} \]

- Let \( \ell \) be the constant of the context free languages pumping lemma.
- Consider \( w = a^p \), where \( p \) is the smallest prime number > \( \ell \).
- If we break \( w = uvxyz \):
  - Let \( |vy| = k, k > 1 \), and \( |uxz| = r = \ell - k \)
  - \( uv^i xy^i z = a^{r+ik} \), so \( \forall i, r + ik \) must be prime
  - set \( i = r + k + 1 \):
    \[ r + ik = r + kr + k^2 + k = (r + k)(k + 1) \]
CF Pumping Lemma

$L = \{ a^p : p \text{ is prime} \}$ is not context free

- Let $\ell$ be the constant of the context free languages pumping lemma.
- Consider $w = a^p$, where $p$ is the smallest prime number $> \ell$.
- If we break $w = uvxyz$:
  - Let $|vy| = k$, $k > 1$, and $|uxz| = r = \ell - k$
  - $uv^i xy^i z = a^{r+ik}$, so $\forall i$, $r + ik$ must be prime
  - Set $i = r + k + 1$:
    - $r + ik = r + kr + k^2 + k = (r + k)(k + 1)$
  - $uv^i xy^i z$ is not in $L$ for $i = r + k + 1$;
CF Pumping Lemma

$L = \{ a^p : p \text{ is prime} \}$ is not context free

- Let $\ell$ be the constant of the context free languages pumping lemma.

- Consider $w = a^p$, where $p$ is the smallest prime number $> \ell$.

- If we break $w = uvxyz$:
  - Let $|vy| = k$, $k > 1$, and $|uxz| = r = \ell - k$
  - $uv^i xy^i z = a^{r+ik}$, so $\forall i$, $r + ik$ must be prime
  - Set $i = r + k + 1$
    - $r + ik = r + kr + k^2 + k = (r + k)(k + 1)$
  - $uv^i xy^i z$ is not in $L$ for $i = r + k + 1$
  - So $L$ is not context-free.
CF Closure Properties

Are the Context-Free Languages closed under union?
CF Closure Properties

- Are context free languages closed under union?
CF Closure Properties

- Are context free languages closed under union? YES!
CF Closure Properties

- Are context free languages closed under union?
  - YES!
  - Proof: Suppose $M_1$ is a PDA accepting $L_1$, and $M_2$ is a PDA accepting $L_2$. 
CF Closure Properties

- Are context free languages closed under union?
  - YES!
  - Proof: Suppose $M_1$ is a PDA accepting $L_1$, and $M_2$ is a PDA accepting $L_2$.
  - Construct a new PDA, $M$, that on first step non-deterministically branches into start state of either $M_1$ or $M_2$. Then, on each branch, acts like original machine.
CF Closure Properties

- Are context free languages closed under union?
  - YES!
  - Proof: Suppose $M_1$ is a PDA accepting $L_1$, and $M_2$ is a PDA accepting $L_2$.
  - Construct a new PDA, $M$, that on first step non-deterministically branches into start state of either $M_1$ or $M_2$. Then, on each branch, acts like original machine.
  - This $M$ accepts $L_1 \cup L_2$. 

Slides modified by Benny Chor, based on original slides by David Galles, Univ. of San Francisco, and Maurice Herlihy, Brown Univ. – p.20
CF Closure Properties

Are context free languages closed under union?

YES!

Proof: Suppose $M_1$ is a PDA accepting $L_1$, and $M_2$ is a PDA accepting $L_2$.

Construct a new PDA, $M$, that on first step non-deterministically branches into start state of either $M_1$ or $M_2$. Then, on each branch, acts like original machine.

This $M$ accepts $L_1 \cup L_2$. 
CF Closure Properties

Are the context free languages context free languages closed under intersection?
CF Closure Properties

- Are the context free languages context free languages closed under intersection?
- Hint – can we intersect two context free languages languages to get $0^n1^n2^n$?
CF Closure Properties

Are the context free languages closed under intersection?
CF Closure Properties

Are the context free languages closed under intersection?

\[ S_1 \rightarrow A_1 B_1 \]
\[ A_1 \rightarrow 0A_1 1|01 \]
\[ B_1 \rightarrow 2B_1|\varepsilon \]

\[ S_2 \rightarrow A_2 B_2 \]
\[ A_2 \rightarrow 0A_2|\varepsilon \]
\[ B_2 \rightarrow 1B_2 2|12 \]

\[ L_1 = 0^n 1^n 2^* \]
\[ L_2 = 0^* 1^n 2^n \]
CF Closure Properties

- Are the context free languages closed under intersection?

\[ S_1 \rightarrow A_1B_1 \]
\[ A_1 \rightarrow 0A_11|01 \]
\[ B_1 \rightarrow 2B_1|\varepsilon \]
\[ S_2 \rightarrow A_2B_2 \]
\[ A_2 \rightarrow 0A_2|\varepsilon \]
\[ B_2 \rightarrow 1B_22|12 \]

\[ L_1 = 0^n1^n2^* \]
\[ L_2 = 0^*1^n2^n \]

\[ L_1 \cap L_2 = 0^n1^n2^n \]
CF Closure Properties

- Are the context free languages closed under intersection?

\[ S_1 \to A_1 B_1 \quad S_2 \to A_2 B_2 \]
\[ A_1 \to 0A_1 1|01 \quad A_2 \to 0A_2|\varepsilon \]
\[ B_1 \to 2B_1|\varepsilon \quad B_2 \to 1B_2 2|12 \]

\[ L_1 = 0^n1^n2^n \quad L_2 = 0^*1^n2^n \]

\[ L_1 \cap L_2 = 0^n1^n2^n \]

- \( L_1 \) is a context free language, \( L_2 \) is a context free language, but \( L_1 \cap L_2 \) is not a context free languages
CF Closure Properties

- Are the context free languages closed under intersection?

\[
S_1 \rightarrow A_1 B_1 \\
A_1 \rightarrow 0A_1 1|01 \\
B_1 \rightarrow 2B_1|\varepsilon \\
S_2 \rightarrow A_2 B_2 \\
A_2 \rightarrow 0A_2|\varepsilon \\
B_2 \rightarrow 1B_2 2|12
\]

\[
L_1 = 0^n1^n2^* \\
L_2 = 0^*1^n2^n
\]

\[
L_1 \cap L_2 = 0^n1^n2^n
\]

- \(L_1\) is a context free language, \(L_2\) is a context free language, but \(L_1 \cap L_2\) is not a context free languages
CF Closure Properties

Are the context free languages context free languages closed under intersection with a regular language?
CF Closure Properties

- Are the context free languages context free languages closed under intersection with a regular language?

- That is, if $L_1$ is context free languages, and $L_2$ is regular, must $L_1 \cap L_2$ be context free languages?
CF Closure Properties

Is $L = \{(0 + 1 + 2)^* : \text{# of 0's} = \text{# of 1's} = \text{# of 2's}\}$ context free?
CF Closure Properties

\[ L \triangleq \{(0 \cup 1 \cup 2)^* : \#0's = \#1's = \#2's \} \]
CF Closure Properties

- $L \triangleq \{(0 \cup 1 \cup 2)^* : \# \text{0’s} = \# \text{1’s} = \# \text{2’s}\}$

- Is $L$ context free?
CF Closure Properties

- \( L \triangleq \{ (0 \cup 1 \cup 2)^* : \# \ 0 \text{'s} = \# \ 1 \text{'s} = \# \ 2 \text{'s} \} \)

- Is \( L \) context free?
  - \( L \cap 00^*11^*22^* = \{ 0^n1^n2^n : n > 0 \} \) which is not context free.
CF Closure Properties

- \( L \triangleq \{(0 \cup 1 \cup 2)^* : \# 0's = \# 1's = \# 2's \} \)

- Is \( L \) context free?
  - \( L \cap 00^*11^*22^* = \{0^n1^n2^n : n > 0\} \) which is not context free.
  - Context free languages intersected with a regular languages are context free
CF Closure Properties

- \( L \triangleq (0 \cup 1 \cup 2)^* : \# 0\text{'s} = \# 1\text{'s} = \# 2\text{'s} \)

Is \( L \) context free?

- \( L \cap 00^*11^*22^* = \{0^n1^n2^n : n > 0\} \) which is not context free.
- Context free languages intersected with a regular languages are context free
- \( 00^*11^*22^* \) is regular
CF Closure Properties

- \( L \triangleq \{ (0 \cup 1 \cup 2)^* : \# \text{0's} = \# \text{1's} = \# \text{2's} \} \)

- Is \( L \) context free?
  - \( L \cap 00^*11^*22^* = \{0^n1^n2^n : n > 0\} \) which is not context free.

  Context free languages intersected with a regular languages are context free
  - \( 00^*11^*22^* \) is regular
  - So \( L \) is not a context free language
Algorithmic Questions Regarding DFAs

Given a regular expression, $R$, find the smallest DFA (minimum number of states) that accepts $L(R)$.

Initial Idea: Use the algorithm describe in class to transform $R$ into an NFA. Then transform this NFA into a DFA, $M$. 
Algorithmic Questions Regarding DFAs

Given a regular expression, $R$, find the smallest $DFA$ (minimum number of states) that accepts $L(R)$.

- Initial Idea: Use the algorithm describe in class to transform $R$ into an $NFA$. Then transform this $NFA$ into a $DFA$, $M$.
- That’s very nice, but how do we know $M$ is smallest?
Algorithmic Questions Regarding DFAs

Given a regular expression, $R$, find the smallest DFA (minimum number of states) that accepts $L(R)$.

- Initial Idea: Use the algorithm describe in class to transform $R$ into an NFA. Then transform this NFA into a DFA, $M$.
- That’s very nice, but how do we know $M$ is smallest?
- We don’t!
Algorithmic Questions for DFAs (2)

Given a regular expression, $R$, find the smallest DFA that accepts $L(R)$ (minimum number of states).

But we can enumerate all DFAs that are strictly smaller than $M$. 
Algorithmic Questions for DFAs (2)

Given a regular expression, $R$, find the smallest DFA that accepts $L(R)$ (minimum number of states).

- But we can enumerate all DFAs that are strictly smaller than $M$.
- For each such $M_i$, test if $L(M_i) = L(M)$ (we saw an algorithm for this).
Algorithmic Questions for DFAs (2)

Given a regular expression, $R$, find the smallest DFA that accepts $L(R)$ (minimum number of states).

- But we can enumerate all DFAs that are strictly smaller than $M$.
- For each such $M_i$, test if $L(M_i) = L(M)$ (we saw an algorithm for this).
- Take the smallest such $M_i$. 
Algorithmic Questions for DFAs (2)

Given a regular expression, $R$, find the smallest DFA that accepts $L(R)$ (minimum number of states).

- But we can enumerate all DFAs that are strictly smaller than $M$.
- For each such $M_i$, test if $L(M_i) = L(M)$ (we saw an algorithm for this).
- Take the smallest such $M_i$.
- Algorithm is not very efficient. If smallest $M$ has $n$ states, algorithm will take time that is exponential in $n$. 
Algorithmic Questions for DFAs (2)

Given a regular expression, $R$, find the smallest DFA that accepts $L(R)$ (minimum number of states).

- But we can enumerate all DFAs that are strictly smaller than $M$.
- For each such $M_i$, test if $L(M_i) = L(M)$ (we saw an algorithm for this).
- Take the smallest such $M_i$.
- Algorithm is not very efficient. If smallest $M$ has $n$ states, algorithm will take time that is exponential in $n$.
- More efficient algorithm is known, due to Myhill and Nerode.
Algorithmic Questions Regarding CFGs

Given a CFG, $G$, and a string $w$, does $G$ generate $w$?
Algorithmic Questions Regarding CFGs

Given a CFG, $G$, and a string $w$, does $G$ generate $w$?

Initial Idea: Design an algorithm that tries all derivations.
Algorithmic Questions Regarding CFGs

Given a CFG, $G$, and a string $w$, does $G$ generate $w$?

Initial Idea: Design an algorithm that tries all derivations.

Problem: If $G$ does not generate $w$, we’ll never stop.
Algorithmic Questions for CFGs (2)

**Lemma:** If $G$ is in Chomsky normal form, $|w| = n$, and $w$ is generated by $G$, then $w$ has a derivation of length $2n - 1$ or less.

We won’t prove this (go ahead — try it at home!).
Algorithmic Questions for CFGs (2)

Lemma: If $G$ is in Chomsky normal form, $|w| = n$, and $w$ is generated by $G$, then $w$ has a derivation of length $2n - 1$ or less.

We won’t prove this (go ahead — try it at home!).

Algorithm’s idea:

First, convert $G$ to Chomsky normal form.
Algorithmic Questions for CFGs (2)

**Lemma:** If $G$ is in Chomsky normal form, $|w| = n$, and $w$ is generated by $G$, then $w$ has a derivation of length $2n - 1$ or less.

We won’t prove this (go ahead — try it at home!).

Algorithm’s idea:

- First, convert $G$ to Chomsky normal form.
- Now need only consider a finite number of derivations – those of length $2n - 1$ or less.
Algorithmic Questions for CFGs (2)

**Lemma:** If $G$ is in Chomsky normal form, $|w| = n$, and $w$ is generated by $G$, then $w$ has a derivation of length $2n - 1$ or less.

We won’t prove this (go ahead — try it at home!).

Algorithm’s idea:

1. First, convert $G$ to Chomsky normal form.
2. Now need only consider a finite number of derivations – those of length $2n - 1$ or less.
Algorithmic Questions for CFGs (3)

**Theorem:** There is an algorithm (that halts on every inputs) $A$, that on inputs $G$ and $w$, decides if $G$ generates $w$. 
Algorithmic Questions for CFGs (3)

**Theorem:** There is an algorithm (that halts on every inputs) $A$, that on inputs $G$ and $w$, decides if $G$ generates $w$.

On input $\langle G, w \rangle$, where $G$ is a grammar and $w$ a string,

1. Convert $G$ to Chomsky normal form.
2. List all derivations with $2n - 1$ steps, were $n = |w|$.
3. If any generates $w$, accept, otherwise reject.
Algorithmic Questions for CFGs (4)

**Theorem:** There is an algorithm (that halts on every inputs) $A$, that on inputs $G$ and $w$, decides if $G$ generates $w$. 
Algorithmic Questions for CFGs (4)

**Theorem:** There is an algorithm (that halts on every inputs) $A$, that on inputs $G$ and $w$, decides if $G$ generates $w$.

**Remarks:**

- Related to problem of compiling prog. languages.
Algorithmic Questions for CFGs (4)

**Theorem:** There is an algorithm (that halts on every inputs) $A$, that on inputs $G$ and $w$, decides if $G$ generates $w$.

**Remarks:**
- Related to problem of compiling prog. languages.
- Would you want to use this algorithm at work?
Algorithmic Questions for CFGs (4)

**Theorem:** There is an algorithm (that halts on every inputs) $A$, that on inputs $G$ and $w$, decides if $G$ generates $w$.

**Remarks:**

- Related to problem of compiling prog. languages.
- Would you want to use this algorithm at work?
- Every theorem about CFLs is also about PDAs.
Emptiness of CFGs

Given a CFG, $G$, is $L(G) = \emptyset$?
Emptiness of CFGs

Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there any string $w$, such that $G$ generate $w$?
Emptiness of CFGs

Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there any string $w$, such that $G$ generate $w$?

**Theorem:** There is an algorithm that solves this problem (and always halts).
Emptiness of CFGs

Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there any string $w$, such that $G$ generate $w$?

**Theorem:** There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:
Emptiness of CFGs

Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there any string $w$, such that $G$ generate $w$?

**Theorem:** There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:

**Bad Idea:** We know how to test whether $w \in L(G)$ for any string $w$, so just try it for each $w$. (criticize this...)
Emptiness of CFGs

Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there any string $w$, such that $G$ generate $w$?

**Theorem:** There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:

**Bad Idea:** We know how to test whether $w \in L(G)$ for any string $w$, so just try it for each $w$. (criticize this...)

**Better Idea:** Can the start variable generate a string of terminals?
Emptiness of CFGs

Given a CFG, $G$, is $L(G) = \emptyset$?

In other words, is there any string $w$, such that $G$ generate $w$?

**Theorem:** There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:

**Bad Idea:** We know how to test whether $w \in L(G)$ for any string $w$, so just try it for each $w$. (criticize this...)

**Better Idea:** Can the start variable generate a string of terminals?

**Even Better Idea:** Can a particular variable generate a string of terminals?
CFG Emptiness (2)

Algorithm: On input $G$ (a CFG),
**CFG Emptiness (2)**

Algorithm: On input $G$ (a CFG),

1. Mark all terminal symbols in $G$. 
**CFG Emptiness** (2)

Algorithm: On input $G$ (a CFG),

1. Mark all terminal symbols in $G$.
2. Repeat until no new variables become marked.
CFG Emptiness (2)

Algorithm: On input $G$ (a CFG),

1. Mark all terminal symbols in $G$.
2. Repeat until no new variables become marked.
3. Mark any $A$ where $A \rightarrow U_1 U_2 \ldots U_k$ and all $U_i$ have already been marked.
**CFG Emptiness (2)**

Algorithm: On input $G$ (a CFG),

1. Mark all terminal symbols in $G$.
2. Repeat until no new variables become marked.
3. Mark any $A$ where $A \rightarrow U_1 U_2 \ldots U_k$ and all $U_i$ have already been marked.
4. If start symbol marked, accept, else reject.

♣

Slides modified by Benny Chor, based on original slides by David Galles, Univ. of San Francisco, and Maurice Herlihy, Brown Univ. – p.33
Given a CFG, $G$, is $L(G) = \Sigma^*$?
CFGs “Fullness”

Given a CFG, $G$, is $L(G) = \Sigma^*$?

We just saw an algorithm to determine, given a CFG, $G$, if $L(G) = \emptyset$
CFGs “Fullness”

Given a CFG, $G$, is $L(G) = \Sigma^*$?

We just saw an algorithm to determine, given a CFG, $G$, if $L(G) = \emptyset$

$L(G) = \Sigma^*$ iff $L(G) = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?
CFGs “Fullness”

Given a CFG, $G$, is $L(G) = \Sigma^*$?

We just saw an algorithm to determine, given a CFG, $G$, if $L(G) = \emptyset$

$L(G) = \Sigma^*$ iff $\overline{L(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?

Unfortunately, CFGs are not closed under complement.
**CFGs “Fullness”**

Given a CFG, $G$, is $L(G) = \Sigma^*$?

We just saw an algorithm to determine, given a CFG, $G$, if $L(G) = \emptyset$

$L(G) = \Sigma^*$ iff $\overline{L(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?

Unfortunately, CFGs are not closed under complement.

**Fact:** There is no algorithm to solve this problem.
CFGs “Fullness”

Given a CFG, $G$, is $L(G) = \Sigma^*$?

We just saw an algorithm to determine, given a CFG, $G$, if $L(G) = \emptyset$.

$L(G) = \Sigma^*$ iff $\overline{L(G)} = \emptyset$.

Why not modify the algorithm so it determines emptiness of the complement?

Unfortunately, CFGs are not closed under complement.

Fact: There is no algorithm to solve this problem.

We are not prepared to prove this remarkable fact (yet).
When Are Two CFGs equivalent?

Given two CFGs, $G, H$, is $L(G) = L(H)$?
When Are Two CFGs equivalent?

Given two CFGs, $G, H$, is $L(G) = L(H)$?

Hey, we did this already for equivalence of DFAs!

We constructed $C$ from $A$ and $B$:

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

and tested whether $L(C)$ is empty.
When Are Two CFGs equivalent?

Given two CFGs, $G$, $H$, is $L(G) = L(H)$?

Hey, we did this already for equivalence of DFAs!

We constructed $C$ from $A$ and $B$:

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

and tested whether $L(C)$ is empty.

Stop! Danger! Abyss ahead!
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!

The class of context-free languages is not closed under complementation or intersection.
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!

The class of context-free languages is not closed under complementation or intersection.

Fact: There is no algorithm to solve this problem.
When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!

The class of context-free languages is not closed under complementation or intersection.

Fact: There is no algorithm to solve this problem.

We are not prepared to prove this remarkable fact (yet).
A Short Summary

- Regular Languages $\equiv$ Finite Automata.
A Short Summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
A Short Summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
- Most algorithmic problems for finite automata are solvable.
A Short Summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
- Most algorithmic problems for finite automata are solvable.
- Some algorithmic problems for finite automata are not solvable.
A Short Summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
- Most algorithmic problems for finite automata are solvable.
- Some algorithmic problems for finite automata are not solvable.
- Pumping lemmata for both classes of languages.
A Short Summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
- Most algorithmic problems for finite automata are solvable.
- Some algorithmic problems for finite automata are not solvable.
- Pumping lemmata for both classes of languages.
- There are additional languages out there.
The View Over The Horizon

- Regular
- Context free
- Decidable
- Enumerable
A Finite Automaton

011011001
read  unread
A Pushdown Automaton

0110110
read unread

0110110
read unread

a b b a

pop push

abba

Slides modified by Benny Chor, based on original slides by David Galles, Univ. of San Francisco, and Maurice Herlihy, Brown Univ. – p.40
A Turing Machine
Alan Turing (1912–1954)

http://www.turing.org.uk/turing/index.html
A Turing Machine (TM)

- uses infinite tape for its memory.
A Turing Machine (TM)

- uses infinite tape for its memory.
- at any point in a computation, only a finite portion of tape has been accessed.
A Turing Machine (TM)

- uses infinite tape for its memory.
- at any point in a computation, only a finite portion of tape has been accessed.
- tape initially contains input string, followed by blanks (\_s).
A Turing Machine (TM)

- uses infinite tape for its memory.
- at any point in a computation, only a finite portion of tape has been accessed.
- tape initially contains input string, followed by blanks (\(\_\)s).
- machine can read from and write on tape (storage device).
A Turing Machine (TM)

- uses infinite tape for its memory.
- at any point in a computation, only a finite portion of tape has been accessed.
- tape initially contains input string, followed by blanks (\_s).
- machine can read from and write on tape (storage device).
- machine can move its “head” both left and right on tape.
A Turing Machine (TM)

- uses infinite tape for its memory.
- at any point in a computation, only a finite portion of tape has been accessed.
- tape initially contains input string, followed by blanks (\_s).
- machine can read from and write on tape (storage device).
- machine can move its “head” both left and right on tape.
- machine halts in either accept or reject states.
A Turing Machine (TM)

- uses infinite tape for its memory.
- at any point in a computation, only a finite portion of tape has been accessed.
- tape initially contains input string, followed by blanks (s).
- machine can read from and write on tape (storage device).
- machine can move its “head” both left and right on tape.
- machine halts in either accept or reject states.
- machine can also run forever, never halting.
TM vs. DFA: Differences

A Turing machine can both write on the tape and read from it.
TM vs. DFA: Differences

- A Turing machine can both write on the tape and read from it.
- The read-write head can move both to the left and to the right.
TM vs. DFA: Differences

- A Turing machine can both write on the tape and read from it.
- The read-write head can move both to the left and to the right
- the tape is infinite
TM vs. DFA: Differences

- A Turing machine can both write on the tape and read from it.
- The read-write head can move both to the left and to the right
- The tape is infinite
- Special accepting/rejecting states take immediate effect
Example

A machine that tests for membership in the language

\[ A = \{ w\#w \mid w \in \{0, 1\}^* \} \]
Example

A machine that tests for membership in the language

\[ A = \{w#w \mid w \in \{0, 1\}^*\} \]

Zig-zags across tape, crossing off matching symbols.
Example (2)

- tape head starts over leftmost symbol
Example (2)

- Tape head starts over leftmost symbol
- Record symbol in control and overwrite X on tape
Example (2)

- tape head starts over leftmost symbol
- record symbol in control and overwrite X on tape
- scan right: reject if encounter blank (⟲) before #
Example (2)

tape head starts over leftmost symbol
| record symbol in control and overwrite $X$ on tape |
| scan right: reject if encounter blank $\_\_\_\_\_\_$ before $\#$ |
| when $\#$ encountered, move right one space |
Example (2)

- Tape head starts over leftmost symbol.
- Record symbol in control and overwrite $X$ on tape.
- Scan right: reject if encounter blank (▁) before #.
- When # encountered, move right one space.
- If symbols don’t match, reject.
Example (3)

write $X$, replacing current symbol ($0$ or $1$)
Example (3)

- write $X$, replacing current symbol (0 or 1)
- scan left, past # to $X$
Example (3)

- write $X$, replacing current symbol (0 or 1)
- scan left, past $#$ to $X$
- move one space right
Example (3)

- write $X$, replacing current symbol (0 or 1)
- scan left, past # to $X$
- move one space right
- record symbol and writes $X$ in its place
Example (3)

- write $X$, replacing current symbol (0 or 1)
- scan left, past # to $X$
- move one space right
- record symbol and writes $X$ in its place
- scan right past # to $X$ …
Example (4)

finally, scan left
Example (4)

- finally, scan left
- if $X$ encountered, keep going left
Example (4)

- finally, scan left
- if \( X \) encountered, keep going left
- if 0 or 1 encountered, reject
Example (4)

- finally, scan left
- if $X$ encountered, keep going left
- if 0 or 1 encountered, reject
- when blank ( ) encountered, accept
Formal Definition

We focus on the transition function

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

$$\delta(q, a) = (r, b, L)$$ means:

- in state $$q$$ where head reads tape symbol $$a$$,
Formal Definition

We focus on the transition function

\[ \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \].

\[ \delta(q, a) = (r, b, L) \] means:
- in state \( q \) where head reads tape symbol \( a \),
- the machine writes \( b \), replacing the \( a \),
Formal Definition

We focus on the transition function

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}.$$ 

$$\delta(q, a) = (r, b, L)$$ means:

- in state $q$ where head reads tape symbol $a$,
- the machine writes $b$, replacing the $a$,
- enters state $r$, 
Formal Definition

We focus on the transition function

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}.$$ 

$$\delta(q, a) = (r, b, L)$$ means:
- in state $q$ where head reads tape symbol $a$,
- the machine writes $b$, replacing the $a$,
- enters state $r$,
- and moves the head left (this is what the $L$ stands for).
Formal Definition (2)

A Turing machine (TM) is a 7-tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$, where

- $Q$ is a finite set of states,
Formal Definition (2)

A Turing machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

- $Q$ is a finite set of states,
- $\Sigma$ is the input alphabet not containing the blank symbol,
Formal Definition (2)

A Turing machine (TM) is a 7-tuple \((Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)\), where

- \(Q\) is a finite set of states,
- \(\Sigma\) is the input alphabet not containing the blank symbol, \(\square\)
- \(\Gamma\) is the tape alphabet, where \(\square \in \Gamma\) and \(\Sigma \subset \Gamma\).
Formal Definition (2)

A Turing machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

- $Q$ is a finite set of states,
- $\Sigma$ is the input alphabet not containing the blank symbol, $\_\$,
- $\Gamma$ is the tape alphabet, where $\_ \in \Gamma$ and $\Sigma \subset \Gamma$.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
Formal Definition (2)

A Turing machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

- $Q$ is a finite set of states,
- $\Sigma$ is the input alphabet not containing the blank symbol, $\bot$,
- $\Gamma$ is the tape alphabet, where $\bot \in \Gamma$ and $\Sigma \subset \Gamma$.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
Formal Definition (2)

A Turing machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

- $Q$ is a finite set of **states**, 
- $\Sigma$ is the **input alphabet** not containing the blank symbol, $\sqcup$, 
- $\Gamma$ is the **tape alphabet**, where $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$. 
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, 
- $q_0 \in Q$ is the **start state**, 
- $q_a \in Q$ is the **accept state**, and
Formal Definition (2)

A Turing machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

- $Q$ is a finite set of states,
- $\Sigma$ is the input alphabet not containing the blank symbol, $\bot$,
- $\Gamma$ is the tape alphabet, where $\bot \in \Gamma$ and $\Sigma \subset \Gamma$,
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_a \in Q$ is the accept state, and
- $q_r \in Q$ is the reject state.
Formal Definition (3)

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \] computes as follows

- an input of length \( n \), \( w = w_1w_2 \ldots w_n \in \Sigma^* \)
Formal Definition (3)

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \] computes as follows

- an input of length \( n \), \( w = w_1w_2\ldots w_n \in \Sigma^* \)
- is on \( n \) leftmost tape squares
Formal Definition (3)

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \] computes as follows

- an input of length \( n \), \( w = w_1 w_2 \ldots w_n \in \Sigma^* \)
- is on \( n \) leftmost tape squares
- rest of tape contains blanks
Formal Definition (3)

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \] computes as follows

- an input of length \( n \), \( w = w_1 w_2 \ldots w_n \in \Sigma^* \)
- is on \( n \) leftmost tape squares
- rest of tape contains blanks
- read/write head is on leftmost square of tape
Formal Definition (3)

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \] computes as follows

- an input of length \( n \), \( w = w_1 w_2 \ldots w_n \in \Sigma^* \)
- is on \( n \) leftmost tape squares
- rest of tape contains blanks
- read/write head is on leftmost square of tape
- since \( \blank \notin \Sigma \), leftmost blank indicates end of input.
Formal Definition (4)

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$.

When computation starts,

$M$ proceeds according to transition function $\delta$. 
Formal Definition (4)

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$.

When computation starts,

- $M$ proceeds according to transition function $\delta$.
- If $M$ tries to move head beyond left-hand-end of tape, it doesn’t move.
Formal Definition (4)

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$.

When computation starts,

- $M$ proceeds according to transition function $\delta$.
- If $M$ tries to move head beyond left-hand-end of tape, it doesn’t move.
- Computation continues until $q_a$ or $q_r$ is reached,
Formal Definition (4)

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$.

When computation starts,

- $M$ proceeds according to transition function $\delta$.
- If $M$ tries to move head beyond left-hand-end of tape, it doesn’t move.
- Computation continues until $q_a$ or $q_r$ is reached,
- otherwise $M$ runs forever.
Configurations

One step of computation changes

- current state,
Configurations

One step of computation changes

- current state,
- current head position,
Configurations

One step of computation changes
- current state,
- current head position,
- and tape contents.
Configurations

One step of computation changes

- current state,
- current head position,
- and tape contents.

For example, configuration $1011q_70111$ means:
Configurations

One step of computation changes
- current state,
- current head position,
- and tape contents.

For example, configuration $1011q_70111$ means:
- Current state is $q_7$, 
Configurations

One step of computation changes

- current state,
- current head position,
- and tape contents.

For example, configuration $1011q_70111$ means:

- Current state is $q_7$,
- left hand side of tape is $1011$, 
Configurations

One step of computation changes

- current state,
- current head position,
- and tape contents.

For example, configuration $1011q_70111$ means:

- Current state is $q_7$,
- left hand side of tape is $1011$,
- right hand side of tape is $0111$, 
Configurations

One step of computation changes

- current state,
- current head position,
- and tape contents.

For example, configuration $\begin{array}{c}1011q_70111\end{array}$ means:

- Current state is $q_7$,
- left hand side of tape is 1011,
- right hand side of tape is 0111,
- and head is on right hand side 0.
Configurations (2)

If $\delta(q_i, b) = (q_j, c, L)$ then configuration $u a q_i b v$ yields configuration $u q_j a c v$. 
Configurations (2)

- If $\delta(q_i, b) = (q_j, c, L)$ then configuration $u a q_i b v$ yields configuration $u q_j a c v$.
- If $\delta(q_i, b) = (q_j, c, R)$, then configuration $u a q_i b v$ yields configuration $u a c q_j v$.
Configurations (2)

- If $\delta(q_i, b) = (q_j, c, L)$ then configuration $u_a q_i b v$ yields configuration $u q_j a c v$.

- If $\delta(q_i, b) = (q_j, c, R)$, then configuration $u a q_i b v$ yields configuration $u a c q_j v$.

- Special case (1): When head is at left end and tries to move left, it changes state and writes on tape but does not move, so if $\delta(q_i, b) = (q_j, c, L)$, configuration $q_i b v$ yields $q_j c v$. 

Slides modified by Benny Chor, based on original slides by David Galles, Univ. of San Francisco, and Maurice Herlihy, Brown Univ. – p.54
Configurations (2)

- If $\delta(q_i, b) = (q_j, c, L)$ then configuration $u a q_i b v$ yields configuration $u q_j a c v$.

- If $\delta(q_i, b) = (q_j, c, R)$, then configuration $u a q_i b v$ yields configuration $u a c q_j v$.

- Special case (1): When head is at left end and tries to move left, it changes state and writes on tape but does not move, so if $\delta(q_i, b) = (q_j, c, L)$, configuration $q_i b v$ yields $q_j c v$.

- Special case (2): What happens when head is at right end? We let $w q_i$ and $w q_i \square$ denotes the same configuration, so moves to the right can now be accommodated.
More Configurations

We have

- starting configuration $q_0w$
More Configurations

We have

- starting configuration $q_0 w$
- accepting configuration $w_0 q_a w_1$
More Configurations

We have

- starting configuration $q_0 w$
- accepting configuration $w_0 q_a w_1$
- rejecting configuration $w_0 q_r w_1$
More Configurations

We have

- starting configuration \( q_0w \)
- accepting configuration \( w_0q_aw_1 \)
- rejecting configuration \( w_0qw_1 \)
- halting configurations \( w_0q_aw_1 \) and \( w_0qw_1 \)