Foundations of Computing

Lecture 10

Arkady Yerukhimovich

February 15, 2024

Outline

1. Lecture 9 Review
2. CFG == PDA
3. The CFL Pumping Lemma
4. Using the CFL Pumping Lemma

Lecture 9 Review

- Context-Free Grammars
  - Strings generated by grammars
  - Building CFGs
  - Parse Trees

Lecture 9 Review

- Context-Free Grammars
  - Strings generated by grammars
  - Building CFGs
  - Parse Trees

Today

Connect CFGs and PDAs and look at their limitations

Outline

Main Theorem

Theorem

A language is context free (i.e., is generated by a CFG) if an only if some pushdown automaton accepts it.

Main Theorem

Theorem

A language is context free (i.e., is generated by a CFG) if an only if some pushdown automaton accepts it.

Proof:

We need to prove both directions:

Main Theorem

Theorem

A language is context free (i.e., is generated by a CFG) if an only if some pushdown automaton accepts it.

Proof:

We need to prove both directions:

1. If a language is context free, then some PDA accepts it

Main Theorem

Theorem

A language is context free (i.e., is generated by a CFG) if an only if some pushdown automaton accepts it.

Proof:

We need to prove both directions:

1. If a language is context free, then some PDA accepts it
2. If a language is accepted by a PDA, then it is context free

Proof of CFG $G \to$ PDA M

Idea: Construct PDA M s.t. $M(w) = 1$ if there is derivation for w in G

- Recall: Derivation of w in G – sequence of substitutions resulting in w
- Each step gives intermediate string of variables and terminals
- M decides if $\exists$ sequence of substitutions in G leads from start to w

Proof of CFG $G \rightarrow$ PDA M

Idea: Construct PDA M s.t. $M(w) = 1$ if there is derivation for w in G

- Recall: Derivation of w in G – sequence of substitutions resulting in w
- Each step gives intermediate string of variables and terminals
- M decides if $\exists$ sequence of substitutions in G leads from start to w

Algorithm for M:

Proof of CFG $G \to$ PDA M

Idea: Construct PDA M s.t. $M(w) = 1$ if there is derivation for w in G

- Recall: Derivation of w in G – sequence of substitutions resulting in w
- Each step gives intermediate string of variables and terminals
- M decides if $\exists$ sequence of substitutions in G leads from start to w

Algorithm for M:

- M pushes the start variable on its stack

Proof of CFG $G \to$ PDA M

Idea: Construct PDA M s.t. $M(w) = 1$ if there is derivation for w in G

- Recall: Derivation of w in G – sequence of substitutions resulting in w
- Each step gives intermediate string of variables and terminals
- M decides if $\exists$ sequence of substitutions in G leads from start to w

Algorithm for M:

- M pushes the start variable on its stack
- M repeatedly makes substitutions according to G, storing intermediate strings on stack

Proof of CFG $G \to$ PDA M

Idea: Construct PDA M s.t. $M(w) = 1$ if there is derivation for w in G

- Recall: Derivation of w in G – sequence of substitutions resulting in w
- Each step gives intermediate string of variables and terminals
- M decides if $\exists$ sequence of substitutions in G leads from start to w

Algorithm for M:

- M pushes the start variable on its stack
- M repeatedly makes substitutions according to G, storing intermediate strings on stack
- $M(w) = 1$ if some intermediate string equals w

Proof of CFG $G \to$ PDA M

Idea: Construct PDA M s.t. $M(w) = 1$ if there is derivation for w in G

- Recall: Derivation of w in G – sequence of substitutions resulting in w
- Each step gives intermediate string of variables and terminals
- M decides if $\exists$ sequence of substitutions in G leads from start to w

Algorithm for M:

- M pushes the start variable on its stack
- M repeatedly makes substitutions according to G, storing intermediate strings on stack
- $M(w) = 1$ if some intermediate string equals w

Challenges

1. May be many substitution rules at each step, how do we choose one?
2. How does M store the intermediate strings?

Proof of CFG $G \rightarrow$ PDA M

Challenges

1. May be many substitution rules at each step, how do we choose one?
2. How does M store the intermediate strings?

Solutions:

Proof of CFG $G \to$ PDA M

Challenges

1. May be many substitution rules at each step, how do we choose one?
2. How does M store the intermediate strings?

Solutions:

1. Rely on non-determinism of M to choose correct substitution rule

Proof of CFG $G \rightarrow$ PDA M

Challenges

1. May be many substitution rules at each step, how do we choose one?
2. How does M store the intermediate strings?

Solutions:

1. Rely on non-determinism of M to choose correct substitution rule
2. Idea: Just store the strings on the stack

Proof of CFG $G \to$ PDA M

Challenges

1. May be many substitution rules at each step, how do we choose one?
2. How does M store the intermediate strings?

Solutions:

1. Rely on non-determinism of M to choose correct substitution rule
2. Idea: Just store the strings on the stack
   Problem:
   - Need to find variable A to replace, but can only access top symbol.

a a A b b c

Proof of CFG $G \rightarrow$ PDA M

Challenges

1. May be many substitution rules at each step, how do we choose one?
2. How does M store the intermediate strings?

Solutions:

1. Rely on non-determinism of M to choose correct substitution rule
2. Idea: Just store the strings on the stack
   Problem:
   - Need to find variable A to replace, but can only access top symbol.
   - Need to remove any leading terminal characters to get to A

Proof of CFG $G \rightarrow$ PDA M

Challenges

1. May be many substitution rules at each step, how do we choose one?
2. How does M store the intermediate strings?

Solutions:

1. Rely on non-determinism of M to choose correct substitution rule
2. Idea: Just store the strings on the stack
   Problem:
   - Need to find variable A to replace, but can only access top symbol.
   - Need to remove any leading terminal characters to get to A
   - But, if we throw these away, can't tell if they match w

Proof of CFG $G \to$ PDA M

Problem:

- Need to find variable A to replace, but can only access top symbols.
- Need to remove any leading terminal characters to get to A
- But, if we throw these away, can't tell if they match w

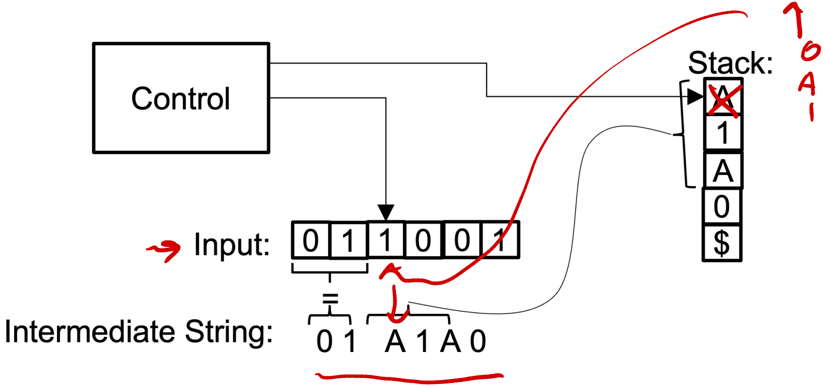
Solution:

Proof of CFG $G \to$ PDA M

Problem:

- Need to find variable A to replace, but can only access top symbols.
- Need to remove any leading terminal characters to get to A
- But, if we throw these away, can't tell if they match w

Solution:



Proof of CFG $G \to$ PDA M

Description of PDA M

1. Push \$ to mark start of stack

Proof of CFG $G \to$ PDA M

Description of PDA M

1. Push \$ to mark start of stack
2. Repeat the following until done
   - If top of stack is variable A, non-deterministically choose a substitution rule and replace A with the right side of rule (push it on stack)

Proof of CFG $G \to$ PDA M

Description of PDA M

1. Push \$ to mark start of stack
2. Repeat the following until done
   - If top of stack is variable A, non-deterministically choose a substitution rule and replace A with the right side of rule (push it on stack)
   - If top of stack is terminal, compare it to next input symbol. If they match, repeat. If not, reject this non-deterministic branch

Proof of CFG $G \rightarrow$ PDA M

Description of PDA M

1. Push \$ to mark start of stack
2. Repeat the following until done
   - If top of stack is variable A, non-deterministically choose a substitution rule and replace A with the right side of rule (push it on stack)
   - If top of stack is terminal, compare it to next input symbol. If they match, repeat. If not, reject this non-deterministic branch
   - If top of stack is \$ symbol, accept if full input has been read

Proof of CFG $G \to$ PDA M

Description of PDA M

1. Push \$ to mark start of stack
2. Repeat the following until done
   - If top of stack is variable A, non-deterministically choose a substitution rule and replace A with the right side of rule (push it on stack)
   - If top of stack is terminal, compare it to next input symbol. If they match, repeat. If not, reject this non-deterministic branch
   - If top of stack is \$ symbol, accept if full input has been read

Picture version of the resulting PDA is in the book

We are done

We are done with this direction of the proof

Proof of PDA $M \rightarrow$ CFG G

Proof of PDA $M \to$ CFG G

Idea: Construct CFG G that generates all strings M accepts

Proof of PDA $M \to$ CFG G

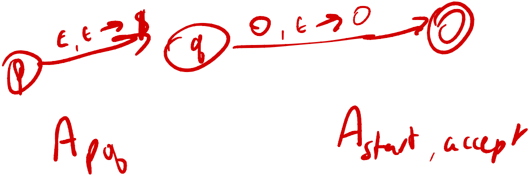
Idea: Construct CFG G that generates all strings M accepts

- G generates strings that cause M to go from start state to an accept state

Proof of PDA $M \to$ CFG G

Idea: Construct CFG G that generates all strings M accepts

- G generates strings that cause M to go from start state to an accept state
- We build something stronger:
  For each pair of states $p, q \in M$, G has a variable A_{pq} such that
  - A_{pq} generates all strings that take M from state p (with an empty stack) to state q (with an empty stack)



Proof of PDA $M \to$ CFG G

Idea: Construct CFG G that generates all strings M accepts

- G generates strings that cause M to go from start state to an accept state
- We build something stronger:
  For each pair of states $p, q \in M$, G has a variable A_{pq} such that
  - A_{pq} generates all strings that take M from state p (with an empty stack) to state q (with an empty stack)

Observations:

Proof of PDA $M \to$ CFG G

Idea: Construct CFG G that generates all strings M accepts

- G generates strings that cause M to go from start state to an accept state
- We build something stronger:
  For each pair of states $p, q \in M$, G has a variable A_{pq} such that
  - A_{pq} generates all strings that take M from state p (with an empty stack) to state q (with an empty stack)

Observations:

- Strings generated by A_{pq} take M from p to q without modifying the stack

Proof of PDA $M \rightarrow$ CFG G

Idea: Construct CFG G that generates all strings M accepts

- G generates strings that cause M to go from start state to an accept state
- We build something stronger:
  For each pair of states $p, q \in M$, G has a variable A_{pq} such that
  - A_{pq} generates all strings that take M from state p (with an empty stack) to state q (with an empty stack)

Observations:

- Strings generated by A_{pq} take M from p to q without modifying the stack
- Thus, $A_{q_0 q_{accept}}$ generates all strings $w \in L(M)$

Proof of PDA $M \to$ CFG G: Building A_{pq}

Assume that M has the following properties:

1. Only one accept state: q_{accept}
2. M empties its stack before accepting
3. All transitions either have form $x, \epsilon \to a$ (push an item on the stack) or $x, a \to \epsilon$ (pop an item off the stack), but not both.

We've already shown how to turn any PDA M into one satisfying these properties

Proof of PDA $M \rightarrow$ CFG G: Building A_{pq}

Consider x taking M from p to q with empty stack

Proof of PDA $M \rightarrow$ CFG G: Building A_{pq}

Consider x taking M from p to q with empty stack

- M's first move on x must be a push – nothing to pop

Proof of PDA $M \to$ CFG G: Building A_{pq}

Consider x taking M from p to q with empty stack

- M's first move on x must be a push – nothing to pop
- M's last move on x must be a pop – need empty stack

Proof of PDA $M \to$ CFG G: Building A_{pq}

Consider x taking M from p to q with empty stack

- M's first move on x must be a push – nothing to pop
- M's last move on x must be a pop – need empty stack
  Two possibilities:

Proof of PDA $M \to$ CFG G: Building A_{pq}

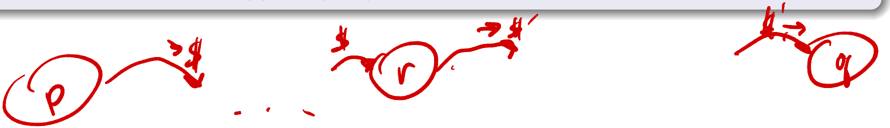
Consider x taking M from p to q with empty stack

- M's first move on x must be a push – nothing to pop
- M's last move on x must be a pop – need empty stack
  Two possibilities:
  - Symbol popped in last step same symbol pushed in first step
    - In this case, stack is only empty at beginning and end
    - Add rule $A_{pq} \to aA_{rs}b$:

Proof of PDA $M \to$ CFG G: Building A_{pq}

Consider x taking M from p to q with empty stack

- M's first move on x must be a push – nothing to pop
- M's last move on x must be a pop – need empty stack
  Two possibilities:
  - Symbol popped in last step same symbol pushed in first step
    - In this case, stack is only empty at beginning and end
    - Add rule $A_{pq} \to aA_{rs}b$:
  - Symbol popped in last step not same symbol pushed in first step
    - Symbol pushed in first step, must be popped before the end, so stack becomes empty at some middle state r
    - Add rule $A_{pq} \to A_{pr}A_{rq}$



Conclusion

We have shown conversions for:

- CFG $G \to$ PDA M, and
- PDA $M \to$ CFG G

Conclusion

We have shown conversions for:

- CFG $G \rightarrow$ PDA M, and
- PDA $M \rightarrow$ CFG G

Takeaway

PDAs recognize exactly the set of context-free languages.

Conclusion

We have shown conversions for:

- CFG $G \to$ PDA M, and
- PDA $M \to$ CFG G

Takeaway

PDAs recognize exactly the set of context-free languages.

Question

Are all languages context-free?

Outline

The CFL Pumping Lemma

Theorem

If L is a CFL, then there exists a pumping length p s.t. for any $s \in L$, with $|s| \geq p$, s can be divided into 5 pieces $s = uvxyz$ satisfying:

1. For each $i \geq 0$, $uv^ixy^iz \in L$
2. $|vy| > 0$
3. $|vxy| \leq p$

The CFL Pumping Lemma

Theorem

If L is a CFL, then there exists a pumping length p s.t. for any $s \in L$, with $|s| \geq p$, s can be divided into 5 pieces $s = uvxyz$ satisfying:

1. For each $i \geq 0$, $uv^ixy^iz \in L$
2. $|vy| > 0$
3. $|vxy| \leq p$

Pumping lemma in math notation:
$\exists p$ s.t $\forall s \in L, |s| \geq p$, $\exists$ partition $s = uvxyz$ s.t. $\forall i, uv^ixy^iz \in L$

The CFL Pumping Lemma

Theorem

If L is a CFL, then there exists a pumping length p s.t. for any $s \in L$, with $|s| \geq p$, s can be divided into 5 pieces $s = uvxyz$ satisfying:

1. For each $i \geq 0$, $uv^ixy^iz \in L$
2. $|vy| > 0$
3. $|vxy| \leq p$

Pumping lemma in math notation:
$\exists p$ s.t $\forall s \in L, |s| \geq p$, $\exists$ partition $s = uvxyz$ s.t. $\forall i, uv^ixy^iz \in L$

Negation of pumping lemma:
$\forall p$, $\exists s \in L, |s| \geq p$ s.t. $\forall$ partitions $s = uvxyz$ $\exists$ i s.t. $uv^ixy^iz \notin L$

Proving the CFL Pumping Lemma (Intuition)

Outline

Using the CFL Pumping Lemma

We use the CFL pumping lemma to prove that L is not a CFL similarly to how we used the regular language pumping lemma.

Using the CFL Pumping Lemma

We use the CFL pumping lemma to prove that L is not a CFL similarly to how we used the regular language pumping lemma.

Specifically:

- Consider the negation:

$$\forall p,\ \exists s \in L, |s| \geq p \text{ s.t. } \forall \text{ partitions } s = uvxyz\ \exists i \text{ s.t. } uv^ixy^iz \notin L$$

Using the CFL Pumping Lemma

We use the CFL pumping lemma to prove that L is not a CFL similarly to how we used the regular language pumping lemma.

Specifically:

- Consider the negation:

$$\forall p,\ \exists s \in L, |s| \geq p \text{ s.t. } \forall \text{ partitions } s = uvxyz\ \exists i \text{ s.t. } uv^i xy^i z \notin L$$

- So, we need to find such an s and prove that for any way to partition it, it cannot be pumped

The Proof Procedure

To use the pumping lemma to prove that L is not CFL, we do the following:

The Proof Procedure

To use the pumping lemma to prove that L is not CFL, we do the following:

1. Assume that L is CFL

The Proof Procedure

To use the pumping lemma to prove that L is not CFL, we do the following:

1. Assume that L is CFL
2. Use pumping lemma to guarantee pumping length p, s.t. all s with $|s| > p$ can be pumped

The Proof Procedure

To use the pumping lemma to prove that L is not CFL, we do the following:

1. Assume that L is CFL
2. Use pumping lemma to guarantee pumping length p, s.t. all s with $|s| > p$ can be pumped
3. Pick some $s \in L$ with $|s| \geq p$

The Proof Procedure

To use the pumping lemma to prove that L is not CFL, we do the following:

1. Assume that L is CFL
2. Use pumping lemma to guarantee pumping length p, s.t. all s with $|s| > p$ can be pumped
3. Pick some $s \in L$ with $|s| \geq p$
4. Demonstrate that s cannot be pumped
    - For each possible division $w = uvxyz$ (with $|vy| > 0$ and $|vxy| \leq p$), find an integer i such that $uv^ixy^iz \notin L$

The Proof Procedure

To use the pumping lemma to prove that L is not CFL, we do the following:

1. Assume that L is CFL
2. Use pumping lemma to guarantee pumping length p, s.t. all s with $|s| > p$ can be pumped
3. Pick some $s \in L$ with $|s| \geq p$
4. Demonstrate that s cannot be pumped
    - For each possible division $w = uvxyz$ (with $|vy| > 0$ and $|vxy| \leq p$), find an integer i such that $uv^ixy^iz \notin L$
5. Contradiction!!!

Example 1

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$, prove L is not CFL

Example 1

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length

Example 1

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Choose $s = a^p b^p c^p \in L$

Example 1

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Choose $s = a^p b^p c^p \in L$
3. By pumping lemma, $s = uvxyz$ s.t. $uv^i xy^i z \in L$ for all i
4. Complete proof by considering all possible values for v, y

Example 1

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Choose $s = a^p b^p c^p \in L$
3. By pumping lemma, $s = uvxyz$ s.t. $uv^i xy^i z \in L$ for all i
4. Complete proof by considering all possible values for v, y
    - v and y both have only one type of symbol (e.g., $v = a^\ell$ and $y = b^{\ell'}$) then $uv^i xy^i z$ has more a's and b's than c's, so is not in L

Example 1

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Choose $s = a^p b^p c^p \in L$
3. By pumping lemma, $s = uvxyz$ s.t. $uv^i xy^i z \in L$ for all i
4. Complete proof by considering all possible values for v, y
    - v and y both have only one type of symbol (e.g., $v = a^{\ell}$ and $y = b^{\ell'}$) then $uv^i xy^i z$ has more a's and b's than c's, so is not in L
    - If either v or y have more than one type of symbol, $uv^i xy^i z$ will have alternating symbols, so not in L

Example 1

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Choose $s = a^p b^p c^p \in L$
3. By pumping lemma, $s = uvxyz$ s.t. $uv^i xy^i z \in L$ for all i
4. Complete proof by considering all possible values for v, y
   - v and y both have only one type of symbol (e.g., $v = a^\ell$ and $y = b^{\ell'}$) then $uv^i xy^i z$ has more a's and b's than c's, so is not in L
   - If either v or y have more than one type of symbol, $uv^i xy^i z$ will have alternating symbols, so not in L
5. Contradiction – Hence L is not CFL

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Try 1: Choose $s = 0^p 1 0^p 1 \in L$

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Try 1: Choose $s = 0^p10^p1 \in L$
3. Try 2: Choose $s = 0^p1^p0^p1^p \in L$

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Try 1: Choose $s = 0^p10^p1 \in L$
3. Try 2: Choose $s = 0^p1^p0^p1^p \in L$
4. Consider all possible cases for vxy ($|vxy| \leq p$)

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Try 1: Choose $s = 0^p 1 0^p 1 \in L$
3. Try 2: Choose $s = 0^p 1^p 0^p 1^p \in L$
4. Consider all possible cases for vxy ($|vxy| \leq p$)
    - vxy does not contain the midpoint of s
        - vxy is left of center – pumping moves a 1 into first character of right half

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Try 1: Choose $s = 0^p10^p1 \in L$
3. Try 2: Choose $s = 0^p1^p0^p1^p \in L$
4. Consider all possible cases for vxy ($|vxy| \leq p$)
    - vxy does not contain the midpoint of s
        - vxy is left of center – pumping moves a 1 into first character of right half
        - vxy is left of center – pumping moves a 0 into last character of left half

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Try 1: Choose $s = 0^p10^p1 \in L$
3. Try 2: Choose $s = 0^p1^p0^p1^p \in L$
4. Consider all possible cases for vxy ($|vxy| \leq p$)
   - vxy does not contain the midpoint of s
     - vxy is left of center – pumping moves a 1 into first character of right half
     - vxy is left of center – pumping moves a 0 into last character of left half
   - vxy does contain the midpoint of s – pumping makes this not match unpumped parts

Example 2

Consider $L = \{ww \mid w \in \{0,1\}^*\}$, prove L is not CFL

Proof:

1. Assume L is CFL, and let p be the pumping length
2. Try 1: Choose $s = 0^p10^p1 \in L$
3. Try 2: Choose $s = 0^p1^p0^p1^p \in L$
4. Consider all possible cases for vxy ($|vxy| \leq p$)
   - vxy does not contain the midpoint of s
     - vxy is left of center – pumping moves a 1 into first character of right half
     - vxy is left of center – pumping moves a 0 into last character of left half
   - vxy does contain the midpoint of s – pumping makes this not match unpumped parts
5. Contradiction – Hence L is not CFL

Exam 1

- This is the end of the material for exam 1
- Next week, review