CS 3313
Foundations of Computing:

Simplification of Context Free Grammars

http://gw-cs3313.github.io

1

Context Free Grammars

- A context free grammar is a grammar G=(V,T,P,S) where all production rules are of the form: $V \rightarrow (V \cup T)^*$
  - Production rules have exactly one variable on the left and a string consisting of variables and terminals on the right.
- Derivations: $\alpha A\beta \Rightarrow \alpha\gamma\beta$ if $A \rightarrow \gamma$ is a production, $\Rightarrow^*$
  - Sentential form: α is in sentential form if $S \Rightarrow^* \alpha$
- Derivation or Parse Trees
  - S is root node, Variables are internal nodes, terminal is leaf node, yield are leaves in pre-order traversal (left to right leaves)
- Equivalence of Parse Trees and Derivations
- If G is a CFG, then L(G), the *language of G*, is

  L(G) = *{w | S =>* w and w is a string over set T}*.

2

Simplification and Parsing: Steps to transform grammars to equivalent more efficient grammar

- 1. Simplification rules: transform a grammar such that:
  - Resulting grammar generates the same language
  - and has "more efficient" production rules in a specific format

Today

- 2. Normal Forms: express all CFGs using a standard "format" for how the production rules are specified
  - Definition of CFGs places no restrictions on RHS of production
  - It is convenient (for parsing algorithms) to restrict to a standard form
    - Chomsky Normal Form (CNF) or Greiback Normal Form (GNF)

Thursday

- 3. Parsing Algorithm: Design a parsing algorithm that takes a grammar in a standard form (CNF) to check if string w is generated by grammar G.

Next week

3

Simplification Rules: Why

- Exhaustive membership (i.e., parsing) algorithm:
  - Input string w of length n.
  - Starting with S, explore all productions for worst case n derivations and determine if it derives string w of length n.
  - How many steps for each of the n derivations:
    - Depends on size of V (set of variables)
    - Depends on size of P (set of productions)

- *Question: do we gain anything if we can remove variables and productions that do not play a part in deriving terminal strings ?*

4

Simplification Rules: Why

- Exhaustive membership (i.e., parsing) algorithm:
  - Input string w of length n.
  - Starting with S, explore all productions for worst case n derivations and determine if it derives string w of length n.
  - How many steps for each of the n derivations:
    - Depends on size of V (set of variables)
    - Depends on size of P (set of productions)

- Observation: if we can remove variables and productions that do not play a part in deriving terminal strings, then we can improve run-time of the algorithm.

5

Comment: a useful algorithm design technique

- There is a family of algorithms that work *inductively*.
- They start discovering some facts that are obvious
  - the basis
- They discover more facts from what they already have discovered
  - induction
- Eventually, nothing more can be discovered, and we are done…..were called *discovery algorithms*

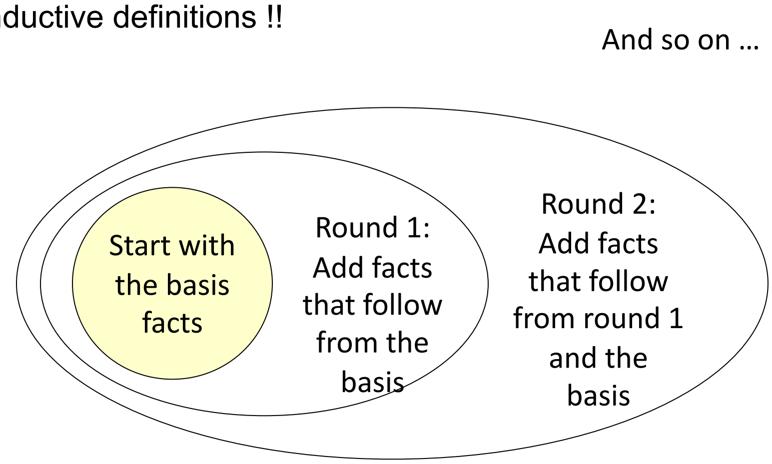
- Observation: decision algorithms for Reg. Lang as well as NFA to DFA (RE to NFA) used this process

6

6

Picture of Discovery

Inductive definitions !!



7

7

CFG Simplification: What is it ?

For a CFG G=(V,T,P,S)

- G has variables/productions that are "*useless*"
  - Variables that cannot derive a terminal string
    - Ex: $B \rightarrow AB$ $S \rightarrow BC$ and $B \rightarrow D$ in the example
  - G has variables that do not appear in any sentential form
    - Variables that cannot be "reached" from start S
- G has *λ– productions* but language does not contain λ
- We can have *unit productions* that create a chain of derivations without contributing at each step to a terminal derivation
  - Ex: $A \rightarrow B$. $B. \rightarrow C$. $C \rightarrow ab$

8

CFG Simplification Process

- **Lemma 2.1**: We derive an equivalent grammar by removing variables that do not derive a terminal string
- **Lemma 2.2**: We can derive an equivalent grammar by removing variables that do not appear in a sentential form
- **Theorem 2.1**: Combine Lemmas 2.1,2.2 to remove useless variables/productions
- **Theorem 2.2**: we can derive an equivalent grammar without λ–productions and get an equivalent grammar
- **Theorem 2.3**: we can derive an equivalent grammar without Unit Productions
- Note: We would like the proofs to result in procedures
  - using iterative algorithms

9

A Substitution Rule

- If A and B are distinct variables, a production of the form $A \rightarrow uBv$ can be replaced by a set of productions in which B is substituted by all strings B derives in one step.
- Consider the grammar

  V = { S, B }, T = { a, b, c }, and productions

  S → a | aaS | abBc   B → abbS | b

- We can replace S → abBc with two productions that replace B (in red), obtaining an equivalent grammar with productions

  S → a | aaS | ababbSc | abbc

  B → abbS | b

10

Useless Variables and Useless Productions

- A variable is *useful* if it occurs in the derivation of at least one string in the language
- A variable is *useless* if:
  - 1. No terminal strings can be derived from the variable
  - 2. The variable symbol cannot be reached from S
  - the variable and any productions in which it appears is considered *useless*

- Ex: Are all variables useful in the grammar below ?

  $S \to A \mid AC$

  $A \to aA \mid \lambda$

  $B \to bA$

  $C \to DC$

11

Lemma 2.1: Removing variables that do not derive terminal strings

- Lemma 2.1: Given a CFG G=(V,T,P,S) we can find an equivalent grammar G_1 =(V', T, P', S) such that for each A in V', there is some w in T* such that A =>* w
  - Every variable in G_1 derives a terminal string
  - $L(G_1) = L(G)$

- We want to construct a proof that leads itself to a (discovery) algorithm
- Proof by induction.

12

Testing Whether a Variable Derives Some Terminal String: Proof

- The inductive proof serves as a *proof of correctness* for the algorithm
- Basis: If there is a production A → w, where w has no variables, then A derives a terminal string.
- Induction: If there is a production A → α, where α consists only of terminals and variables known to derive a terminal string, then A derives a terminal string, i.e., every symbol in α is useful (in Terminals or in V' - useful variables)
- Eventually, we can find no more variables.

13

13

Example: Lemma 2.1 Algorithm to Eliminate Variables that do not derive terminal strings

S -> AB | C   A -> aA | a   B → bB   C → c

V' = Ø (initialize to empty set)

Basis: V' =

14

14

Algorithm to remove variables that do not derive terminal strings: Lemma 2.1

Input: G = (V,T,P,S)

1. $V_{init} := \emptyset$ /* initialize V_{init} to empty set
2. $V' = \{ A \mid A \rightarrow w$ is a production for w in $T^*\}$
3. While V_{init} <> V'
4. $V_{init} = V'$
5. $V' = V_{init} \cup \{A \mid A \rightarrow \alpha$ for some α in $(V_{init} \cup T)^* \}$
6. endwhile

7. $P' = \{ X \rightarrow \alpha \mid X \in V'$ and $\alpha \in (V' \cup T)^* \}$

15

Lemma 2.2: Removing variables or terminals that are not reachable from S

- **Lemma 2.2**: Given a CFG G=(V,T,P,S) we can find an equivalent grammar G_1 =(V', T', P', S) such that for each A in V' U T', there exist α, β in $(V' \cup T')^*$ for which $S \Rightarrow^* \alpha A \beta$
  - Every variable or terminal in G_1 appears in a sentential form
  - i.e., is reachable from S through a series of derivations

- We want to construct a proof that leads itself to a (discovery) algorithm
  - A proof by induction on length of derivation or *use graph properties by constructing a reachability graph*.

16

Example: Lemma 2.2 Algorithm to Eliminate Variables that are not reachable from S

S -> AB A -> aA | a B → bB A → C C → c

D → aEbb| aD E → abb

V’ = ∅ (initialize to empty set)

Basis: V’ =

17

17

Algorithm to remove variables that are not reachable from S: Lemma 2.2

Input: G = (V,T,P,S); Output G’=(V’,T’,P’,S) with all reachable

1. $V_{init} = \emptyset$
2. V’ = { S}
3. While V_{init} <> V’ /* repeat loop until you cannot add more
4. V_{init}= V’
5. V’ = $V_{init} \cup \{X \mid A \to \alpha,\ A \in V_{init}$ and X appears in $\alpha\}$
6. endwhile
7. P’ = { $X \to \alpha \mid X \in V'$ and $\alpha \in (V' \cup T)^*$ }

Simpler approach:
Construct graph, with edge from X to Y where X is LHS of production and Y is on RHS of production
V’ = all nodes reachable from S

18

Example: Procedure for Removing Useless Productions

- Consider the grammar:

  $S \rightarrow AB \mid a$

  $A \rightarrow a$

Apply Lemma 2.2:

Apply Lemma 2.1:

19

Theorem 2.1: Removing useless symbols/productions

- Theorem 2.1: Every nonempty context free language is generated by a CFG G with no useless symbols.

- Proof:

1. Apply Lemma 2.1 and
2. then apply Lemma 2.2
3. Prove by contradiction.

20

Example: Procedure for Removing Useless Productions

- Consider the grammar:

  $S \rightarrow aS \mid A \mid C$

  $A \rightarrow a$ $\quad B \rightarrow aa$ $\quad C \rightarrow aCb$

21

λ-Productions

- A production with λ on the right side is called a *λ-production*
- A variable (symbol) A is called *nullable* if there is a sequence of derivations through which A produces λ : i.e., $A \Rightarrow^* \lambda$
- If a grammar generates a language not containing λ, is there any point in keeping λ-productions ?
- Example: S_1 is nullable

  $S \rightarrow aS_1b$

  $S_1 \rightarrow aS_1b \mid \lambda$
- Is there a grammar without λ-productions that generates same language above?

22

λ-Productions

- A production with λ on the right side is called a *λ-production*
- A variable (symbol) A is called *nullable* if there is a sequence of derivations through which A produces λ
  - A =>* λ
- If a grammar generates a language not containing λ, any λ-productions can be removed
- Example: S_1 is nullable

  $S \rightarrow aS_1b$

  $S_1 \rightarrow aS_1b \mid \lambda$
- Since the language is λ-free, we have the equivalent grammar

  $S \rightarrow aS_1b \mid ab$

  $S_1 \rightarrow aS_1b \mid ab$

23

Example: Nullable Variables

$S \rightarrow AB, \quad A \rightarrow aA \mid \lambda \quad B \rightarrow bB \mid A$

So how do we transform the grammar into
equivalent grammar (minus λ) without any λ productions ?

1. find nullable symbols
2. replace with set of productions

24

24

Theorem 2.2: Removing λ productions

- **Theorem 2.2**: If L = L(G) for some CFG G=(V,T,P,S) then L- {λ } is generated by a CFG G' with no useless symbols or λ productions.

1. First iteratively find *nullable* variables
2. Next replace RHS of production with nullable symbols replaced by λ
3. Then apply algorithms to remove useless symbols (Thm. 2.1)

- Key Idea: turn each production $A \rightarrow X_1 \ldots X_n$ into a set of productions
  - Except, if all X's are nullable (or the body was empty to begin with), do not make a production with λ as the right side
- Proof: formal proof by induction on length of derivation

25

Theorem 2.2: Algorithm to remove λ productions

1. $V_N = \emptyset$ /* these are nullable variables
2. $V_{init} = \{ A \mid A \rightarrow \lambda \}$ /* add A if RHS of prod is λ
3. While V_{init} <> V_N /* repeat loop to add A where A =>*
4. $V_{init} = V_N$
5. $V_N = V_{init} \cup \{A \mid A \rightarrow \alpha$ and α is in $V_{init}^* \}$
6. endwhile
7. Remove all λ productions from P
8. For each production in P, $A \rightarrow \alpha$,

For each $X \in \alpha$ and $X \in V_N$ add productions in which nullable symbols are replaced by λ but not all are replaced by λ

26

Example 1: Removing λ-Productions

$S \rightarrow ABaC$ $\quad A \rightarrow BC$ $\quad B \rightarrow b \mid \lambda$ $\quad C \rightarrow D \mid \lambda$ $\quad D \rightarrow d$

- Nullable symbols =
- New set of productions =

27

Example 2: Eliminating λ-Productions

S -> ABC, A -> aA | λ, B -> bB | λ , C -> λ

- A, B, C, and S are all nullable.
- New grammar:

  S -> ABC | AB | AC | BC | A | B | C

  A -> aA | a

  B -> bB | b

Note: C is now useless.
Eliminate its productions.

28

28

And lastly……Unit-Productions

- A production of the form $A \rightarrow B$ (where A and B are variables) is called a *unit-production*
- Unit-productions add unneeded complexity to a grammar and can usually be removed by simple substitution
- Theorem 2.2 states that any context-free grammar without λ-productions has an equivalent grammar without unit-productions
  - The procedure for eliminating unit-productions assumes that all λ-productions have been previously removed

29

Example: Unit Productions

- S → ASB   A→ C B → b

  C → D D→ a

30

Theorem 2.3: Removing Unit productions

- Theorem 2.3: Every context free language without the empty string is generated by a CFG G=(V,T,P,S) with no useless productions,λ productions, or unit productions.
- Proof:
  - First remove unit productions and then apply Theorem 2.2 and 2.1
- Key idea: If A =>* B by a series of unit productions, and B -> α is a non-unit-production, then add production A -> α.
  - Then, drop all unit productions.
- Formal proof by induction on length of derivation.

31

Algorithm for Removing Unit Productions

1. Draw a dependency graph with an edge from A to B corresponding to every A $\rightarrow$ B production in the grammar
2. Construct a new grammar that includes all the productions from the original grammar, except for the unit-productions
3. Whenever there is a path from A to B in the dependency graph, replace A $\rightarrow$ B with A $\rightarrow$ α using the substitution rule from Lemma 2.0 (but using only the non-unit productions B $\rightarrow$ α in the new grammar)

32

Example: Procedure for Removing Unit-Productions

$S \rightarrow Aa \mid B \qquad A \rightarrow a \mid bc \mid B \qquad B \rightarrow A \mid bb$

- Find all pairs X,Y such that X =>* Y using only unit prod
- Substitute/add new productions

33

Putting it all together: Cleaning Up a Grammar

- **Theorem 2.4:** if L is a CFL, then there is a CFG for L – {λ} that has:
  1. No useless variables (and productions).
  2. No λ-productions.
  3. No unit productions.
- *Theorem 2.4 implies: every string on RHS of prodution is either a single terminal or has length ≥ 2.*

34

34

Cleaning Up CFGs

- Proof: Start with a CFG for L.
- Perform the following steps in order:
  1. Eliminate λ-productions. (Theorem 2.3)
  2. Eliminate unit productions. (Theorem 2.2)
  3. Eliminate variables that derive no terminal string. (Lemma 2.1)
  4. Eliminate variables not reached from the start symbol. (Lemma 2.2)

Must be first. This step can create unit productions or useless variables.

35

35

Next: Procedure to transform any CFG to Chomsky Normal Form

- A CFG is said to be in *Chomsky Normal Form* if every production is of one of these two forms:
  1. A → BC (body is two variables).
  2. A → a (body is a single terminal).
- Theorem: If L is a CFL, then L – { λ} has a CFG in CNF.
  - *Note: Theorem 2.4 implies every string on RHS of prodution is either a single terminal or has length $\geq$ 2.*
    - *This is our starting point when converting to CNF form*
- Question: property of parse trees for CNF grammars ?

36

36

Time to test out the algorithms: Exercise

- Given grammar G=(V,T,P,S), find an equivalent grammar G' with no unit productions, λ productions or useless variables/productions.
  - i.e, clean up the grammar

S → aA | AC | aBB
A → aaA | λ
B → bB | bbC
C → B

37

Normal Forms for Context Free Grammars

- Any context free grammar can be converted to an equivalent grammar in a "normal form"
- Why is this useful ?
- Design a parsing algorithm that assumes a "standard form" for specifying a grammar
  - Becomes part of the program specifications

38

Normal Forms for Context Free Grammars

- Any context free grammar can be converted to an equivalent grammar in a “normal form”
- Chomsky Normal Form (CNF):

  All productions are of the form $A \rightarrow a$ or $A \rightarrow BC$ where a is a terminal symbol and A, B, C are variables
- Greibach Normal Form (GNF):

  All productions are of the form $A \rightarrow a\alpha$ where a is a terminal and α is a string of variables (possibly empty)

39

Chomsky Normal Form

- **Def**: A CFG $G = (V, T, P, S)$ is in Chomsky Normal Form (CNF) if all productions are of the form
  - $A \rightarrow BC$, or
  - $A \rightarrow a$,
- where $A, B, C \in V$, and $a \in T$.
- **Benefit**: Parsing tree for $w \in G$ becomes a binary tree.

40

CNF

- G_1 with production rules:
  - $S \to AS \mid a$
  - $A \to SA \mid b$
- Is G_1 in CNF?

- G_2 with production rules:
  - $S \to AS \mid AAS$
  - $A \to SA \mid aa$
- Is G_2 in CNF?

41

CNF Construction-1

- **Theorem 6.6**: Any CFG $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\widehat{G} = (\widehat{V}, \widehat{T}, S, \widehat{P})$ in CNF.
- *Proof* by constructing $\widehat{G}$ for arbitrary G that has no λ or unit productions [from the simplification algorithms].
  - Note: After simplication all productions are of the form A→a or A→ **α** where | **α**| >= 2
- **Step 1**: Constructing $G_1 = (V_1, T, S, P_1)$ from G by considering all productions P in the form

$$A \to x_1 x_2 \dots x_n$$

where each x_i is either in V or T.

42

CNF Construction-2

- $A \to x_1x_2 \dots x_n$
- If $n = 1$, then x_1 must already be a terminal, since we do not have unit productions.
  - In this case, let P be P_1.
- Otherwise, in V_1, we introduce new variables B_a for each $a \in T$, and $B_a \to a$ is put into P_1.
- Then, for each A, we put into P_1 the production

$$A \to C_1C_2 \dots C_n$$

where $C_i = x_i$ if $x_i \in V$, and $C_i = B_a$ if $x_i = a$.

43

CNF Construction-3

- Part 1 of the algorithm removes all terminals from productions whose RHS has length greater than one, replacing them with newly introduced variables.
- At the end of this step, we have a grammar G_1 with all its productions in the form of either
  - $A \to a$
  - or $A \to C_1C_2 \dots C_n$, where $C_i \in V_1$.

✓ It is easy to see that $L(G_1) = L(G)$.

44

CNF Construction-4

- **Step 2**: Constructing $\hat{G}$ by reducing lengths of the RHS of rules in G_1 when necessary.
- First, from P_1, we put all productions in the form of

  $A \to a$ or $A \to C_1C_2$ into $\hat{P}$.

- For rules with $A \to C_1 \dots C_n, n > 2$, we introduce new variables $D_1, D_2, \dots$ and put into $\hat{P}$ the productions
  - $A \to C_1D_1$
  - $D_1 \to C_2D_2 \dots \dots$
  - $D_{n-1} \to C_{n-1}C_n$, where each $A, D_1, \dots, D_{n-1}$ is in CNF.
- It is easy to see that $\hat{G}$ is in CNF, and $L(\hat{G}) = L(G)$.

45

Summary: Conversion to CNF

- **Theorem 6.6**: Any CFG $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ in CNF.
- **Step 1**: Constructing $G_1 = (V_1, T, S, P_1)$ from G by considering all productions P in the form

  $A \to x_1x_2 \dots x_n$ where each x_i is either in V or T.
  - Add variable V_a and production $V_a \to a$ for each terminal a
  - If x_i is a terminal a, replace with V_a
- **Step 2:** For rules with $A \to C_1 \dots C_n, n > 2$, we introduce new variables $D_1, D_2, \dots$ and put into $\hat{P}$ the productions
  - $A \to C_1D_1$
  - $D_1 \to C_2D_2 \dots \dots$
  - $D_{n-1} \to C_{n-1}C_n$, where each $A, D_1, \dots, D_{n-1}$ is in CNF.

46

CNF Construction-Example

- Consider G with production rules:

$$S \to ABa \quad A \to aab \quad B \to Ac$$

- First of all, no λ or unit or useless productions.
- **Step 1**: For G_1, we add $S \to ABB_a \quad A \to B_aB_aB_b \quad B \to AB_c$ and $B_a \to a \quad B_b \to b \quad B_c \to c$ into P_1.
- **Step 2**: For $\hat{G}$, we add $S \to AD_1 \quad D_1 \to BB_a \quad A \to B_aD_2 \quad D_2 \to B_aB_b \quad B \to AB_c$ and $B_a \to a \quad B_b \to b \quad B_c \to c$ into $\hat{P}$.

47

Example

- P: $S \to ABa \quad A \to aab \quad B \to Ac$
- **Step 1**:

48

Example

- P_1: $S \rightarrow ABB_a$ $A \rightarrow B_aB_aB_b$ $B \rightarrow AB_c$ $B_a \rightarrow a$ $B_b \rightarrow b$ $B_c \rightarrow c$
- **Step 2**:

49

Exercise: CNF Conversion

S → PSQ   P → aPS | a | λ

Q → SbS | P | bb

50