Lecture 7

COT4210 DISCRETE STRUCTURES

DR. MATTHEW B. GERBER

6/14/2016

PORTIONS FROM SIPSER, *INTRODUCTION TO THE THEORY OF COMPUTATION*, 3RD ED., 2013

Some Rules

1. $A \rightarrow$ **0***A***1**
2. $A \rightarrow B$
3. $B \rightarrow$ **#**

These are rules for one thing becoming another

- *A* can become 0*A*1 or *B*
- *B* can only become #

Let's start with *A*. What kinds of strings can we make with these rules?

Some Rules

1. $A \rightarrow$ **0**A**1**
2. $A \rightarrow B$
3. $B \rightarrow$ **#**

These are rules for one thing becoming another

- A can become 0A1 or B
- B can only become #

Let's start with A. What kinds of strings can we make with these rules?

- # *2 then 3*

Some Rules

1. $A \rightarrow 0A1$
2. $A \rightarrow B$
3. $B \rightarrow \#$

These are rules for one thing becoming another

- *A* can become 0*A*1 or *B*
- *B* can only become #

Let's start with *A*. What kinds of strings can we make with these rules?

- # *2 then 3*
- 0#1 *1 then 2 then 3*

Some Rules

1. $A \rightarrow 0A1$
2. $A \rightarrow B$
3. $B \rightarrow \#$

These are rules for one thing becoming another

- *A* can become 0*A*1 or *B*
- *B* can only become #

Let's start with *A*. What kinds of strings can we make with these rules?

- # *2 then 3*
- 0#1 *1 then 2 then 3*
- 00#11 *1, 1, 2, 3*

Some Rules

1. $A \rightarrow$ **0A1**
2. $A \rightarrow B$
3. $B \rightarrow$ **#**

These are rules for one thing becoming another

- *A* can become 0*A*1 or *B*
- *B* can only become #

Let's start with *A*. What kinds of strings can we make with these rules?

- # — *2 then 3*
- 0#1 — *1 then 2 then 3*
- 00#11 — *1, 1, 2, 3*
- 00000#11111 — *1,1,1,1,1,2,3*

Some Rules

1. $A \rightarrow$ **0**A**1**
2. $A \rightarrow B$
3. $B \rightarrow$ **#**

These are rules for one thing becoming another
- *A* can become 0*A*1 or *B*
- *B* can only become #

Let's start with *A*. What kinds of strings can we make with these rules?

#	*2 then 3*
0#1	*1 then 2 then 3*
00#11	*1, 1, 2, 3*
00000#11111	*1,1,1,1,1,2,3*

Do you see how these rules are different from regular languages?

Some Rules

1. $A \rightarrow$ **0**A**1**
2. $A \rightarrow B$
3. $B \rightarrow$ **#**

These are rules for one thing becoming another

r *B*

THEY CAN COUNT!

hat kinds of strings can
rules?

2 then 3

1 then 2 then 3

1, 1, 2, 3

1,1,1,1,1,2,3

Do you see how these rules are different from regular languages?

Context-Free Grammars: In Short

A **context-free grammar** is a series of **substitution rules**, or **productions**

Each rule is a single line

- The symbol on the left is called a **variable**, or **non-terminal** symbol
- The symbols on the right can be any combination of **variables** and **terminal** symbols

There are rules about where symbols can show up

- Only variables can be on the left
- Only *one* variable can be on the left
- Variables *or* terminals can be on the right

1. $A \rightarrow \mathbf{0}A\mathbf{1}$
2. $A \rightarrow B$
3. $B \rightarrow \mathbf{\#}$

Context-Free Grammars: How To Start

We can represent the start symbol in one of two ways

- We can represent it *explicitly*, using the generally-agreed name *S*…

1. $S \rightarrow A$
2. $A \rightarrow \mathbf{0}A\mathbf{1}$
3. $A \rightarrow B$
4. $B \rightarrow \#$

Context-Free Grammars: How To Start

We can represent the start symbol in one of two ways

- We can represent it *explicitly*, using the generally-agreed name *S*…
- Or we can leave it *implicit*, which is more common

If we do not give an explicit start symbol, the start symbol is understood to be the left-hand side of the topmost rule

1. $A \rightarrow \mathbf{0}A\mathbf{1}$
2. $A \rightarrow B$
3. $B \rightarrow$ **#**

Context-Free Grammars: How To Start

We can represent the start symbol in one of two ways

- We can represent it *explicitly*, using the generally-agreed name *S*…
- Or we can leave it *implicit*, which is more common

If we do not give an explicit start symbol, the start symbol is understood to be the left-hand side of the topmost rule

If a start symbol *isn't* the left-hand side of the topmost rule, ***GIVE AN EXPLICIT START SYMBOL!***

- Yes, even if it's obvious
- It's not really obvious enough

1. $A \rightarrow \mathbf{0}A\mathbf{1}$
2. $A \rightarrow B$
3. $B \rightarrow \mathbf{\#}$

Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable

1. $A \rightarrow \mathbf{0}A\mathbf{1}$
2. $A \rightarrow B$
3. $B \rightarrow \#$

____

A

Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable
2. **Find** a variable that's written and a rule that has it as its left-hand side
3. **Replace** the variable with the right-hand side of the rule

1. $A \rightarrow 0A1$
2. $A \rightarrow B$
3. $B \rightarrow \#$

____

0A1

(1)

Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable
2. **Find** a variable that's written and a rule that has it as its left-hand side
3. **Replace** the variable with the right-hand side of the rule
4. **Repeat** from step 2…

1. $A \rightarrow \mathbf{0}A\mathbf{1}$
2. $A \rightarrow B$
3. $B \rightarrow \mathbf{\#}$

___

00A11

(1, 1)

Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable
2. **Find** a variable that's written and a rule that has it as its left-hand side
3. **Replace** the variable with the right-hand side of the rule
4. **Repeat** from step 2…

1. $A \rightarrow \mathbf{0}A\mathbf{1}$
2. $A \rightarrow B$
3. $B \rightarrow \mathbf{\#}$

____

00B11

(1, 1, 2)

Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable
2. **Find** a variable that's written and a rule that has it as its left-hand side
3. **Replace** the variable with the right-hand side of the rule
4. **Repeat** from step 2 until you're out of variables

1. $A \rightarrow 0A1$
2. $A \rightarrow B$
3. $B \rightarrow \#$

00#11

(1, 1, 2, 3)

Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable
2. **Find** a variable that's written and a rule that has it as its left-hand side
3. **Replace** the variable with the right-hand side of the rule
4. **Repeat** from step 2 until you're out of variables

Three things:

- Notice that only terminal symbols remain

1. $A \rightarrow 0A1$
2. $A \rightarrow B$
3. $B \rightarrow \#$

____

00#11

(1, 1, 2, 3)

Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable
2. **Find** a variable that's written and a rule that has it as its left-hand side
3. **Replace** the variable with the right-hand side of the rule
4. **Repeat** from step 2 until you're out of variables

Three things:

- Notice that only terminal symbols remain
- The chain you follow to get to the string is called a **derivation**…

1. $A \rightarrow \mathbf{0}A\mathbf{1}$
2. $A \rightarrow B$
3. $B \rightarrow \mathbf{\#}$

00#11

(1, 1, 2, 3)

$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$

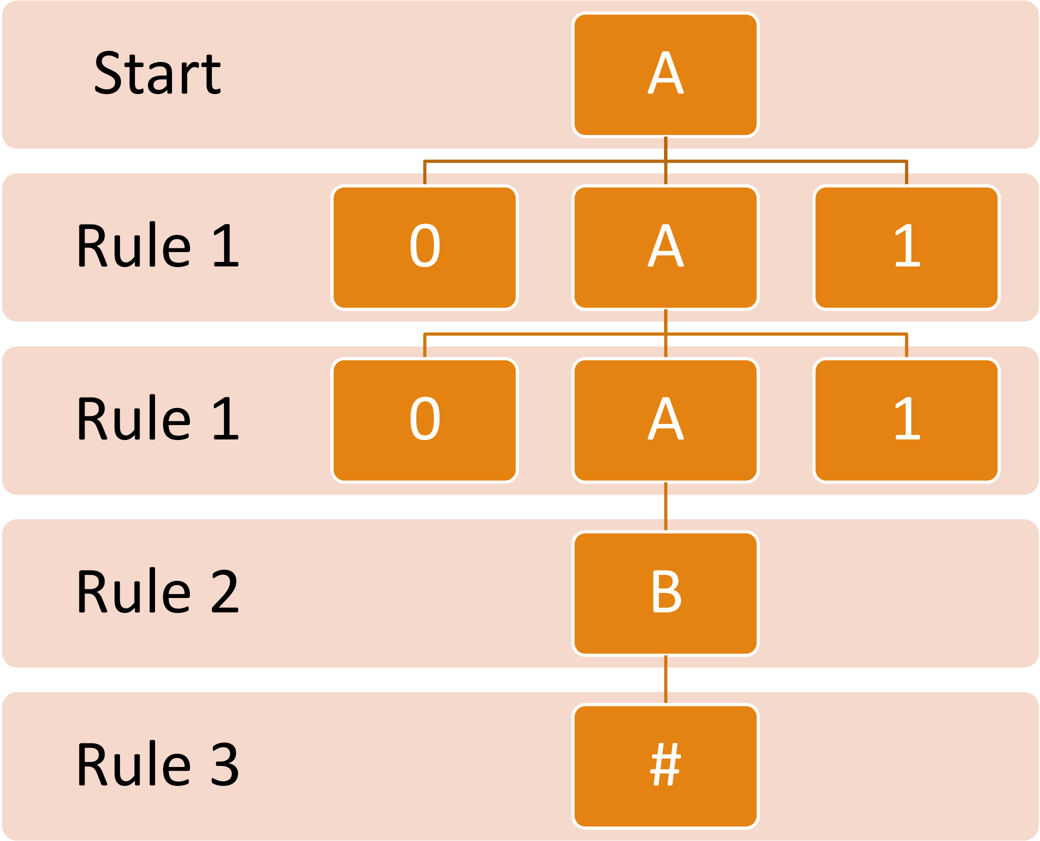
Context-Free Grammars: How To Use

A grammar generates strings

1. Write down the **start** variable
2. **Find** a variable that's written and a rule that has it as its left-hand side
3. **Replace** the variable with the right-hand side of the rule
4. **Repeat** from step 2 until you're out of variables

Three things:

- Notice that only terminal symbols remain
- The chain you follow to get to the string is called a **derivation**…
- …and it can also be represented as a **parse tree**



Another Grammar

1. SENTENCE → ANOUN VERB ANOUN
2. ANOUN → ARTICLE NOUN
3. ARTICLE → **a | an | the**
4. NOUN → **person | eye | music | image**
5. VERB → **hears | sees**

the person hears the music

a person sees the image

an eye sees an image

Another Grammar

1. SENTENCE → ANOUN VERB ANOUN
2. ANOUN → ARTICLE NOUN
3. ARTICLE → **a | an | the**
4. NOUN → **person | eye | music | image**
5. VERB → **hears | sees**

the person hears the music

a person sees the image

an eye sees an image

the image hears an person

a eye hears an music

Context Free Grammars and…

A grammar **generates** strings.

In fact, we can think of the strings that a given grammar can generate as a *set* of strings.

Guess what we call that set of strings.

Go on. Guess.

Definitions: Context-Free Grammar, Context-Free Language

A **context-free grammar** is a 4-tuple $G = (V, \Sigma, R, S)$ where:

- V is a finite set called the ***variables***
- Σ is a finite set, *disjoint* from V, called the ***terminals***
- R is a finite set of ***rules***, each rule allowing a variable to be rewritten as a string of variables and terminals, and
- $S \in V$ is the start variable.

The language L(G) of a grammar is the set of strings that can be generated by that grammar.

A **context-free language** is a language that can be generated by a context-free grammar.

An Arithmetic Example

V = { EXPR, TERM, FACTOR }

Σ = { **a**, +, ×, (,) }

Rules *R* are**…**

EXPR → EXPR + TERM | TERM

TERM → TERM × FACTOR | FACTOR

FACTOR → (EXPR) | a

An Arithmetic Example Generating: **a + a × a**

V = { EXPR, TERM, FACTOR }

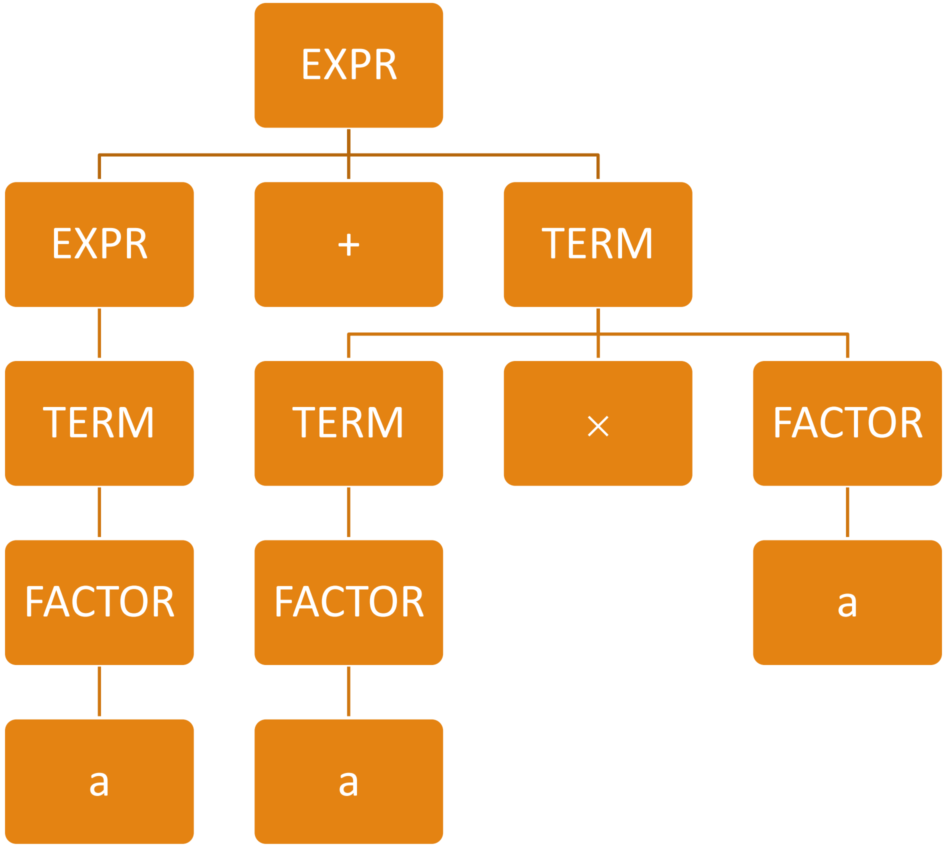
Σ = { **a**, +, ×, (,) }

Rules *R* are…

EXPR → EXPR + TERM | TERM

TERM → TERM × FACTOR | FACTOR

FACTOR → (EXPR) | a



An Arithmetic Example
Generating: **(a + a) × a**

V = { EXPR, TERM, FACTOR }

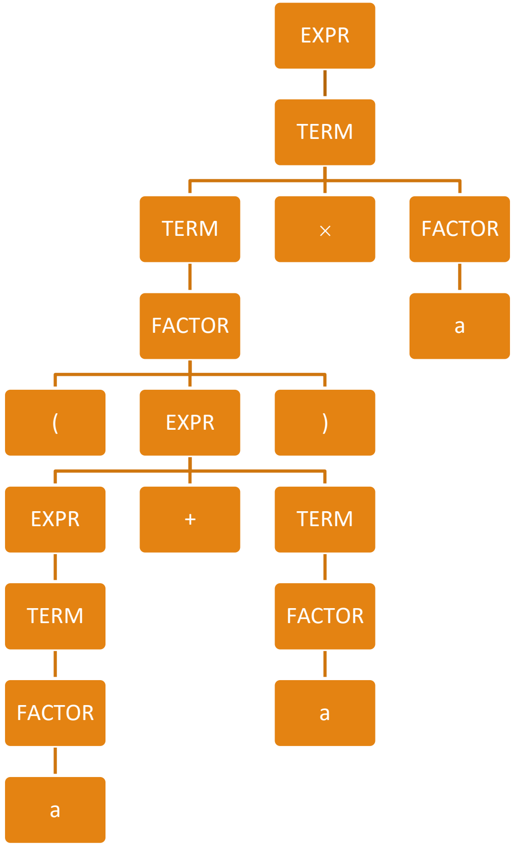
Σ = { **a**, +, ×, (,) }

Rules *R* are...

EXPR → EXPR + TERM | TERM

TERM → TERM × FACTOR | FACTOR

FACTOR → (EXPR) | a



Notes on CFG Design

Divide and conquer applies

- Make simpler portions of the grammar, then make the grammar out of them
- Simulate recursion by letting a variable generate itself, directly or indirectly

DFAs are easy to simulate with CFGs

- Make a variable V_i for each state q_i
- If $\delta(q_i, q_k) = \mathbf{a}$, make a rule $V_i \rightarrow \mathbf{a}V_k$
- If q_i is an accept state, make a rule $V_i \rightarrow \varepsilon$
- Make V_0 the starting variable

Since you can have $V \rightarrow \mathbf{a}V\mathbf{b}$, you can count

Ambiguity

V = { EXPR }

Σ = { **a**, +, ×, (,) }

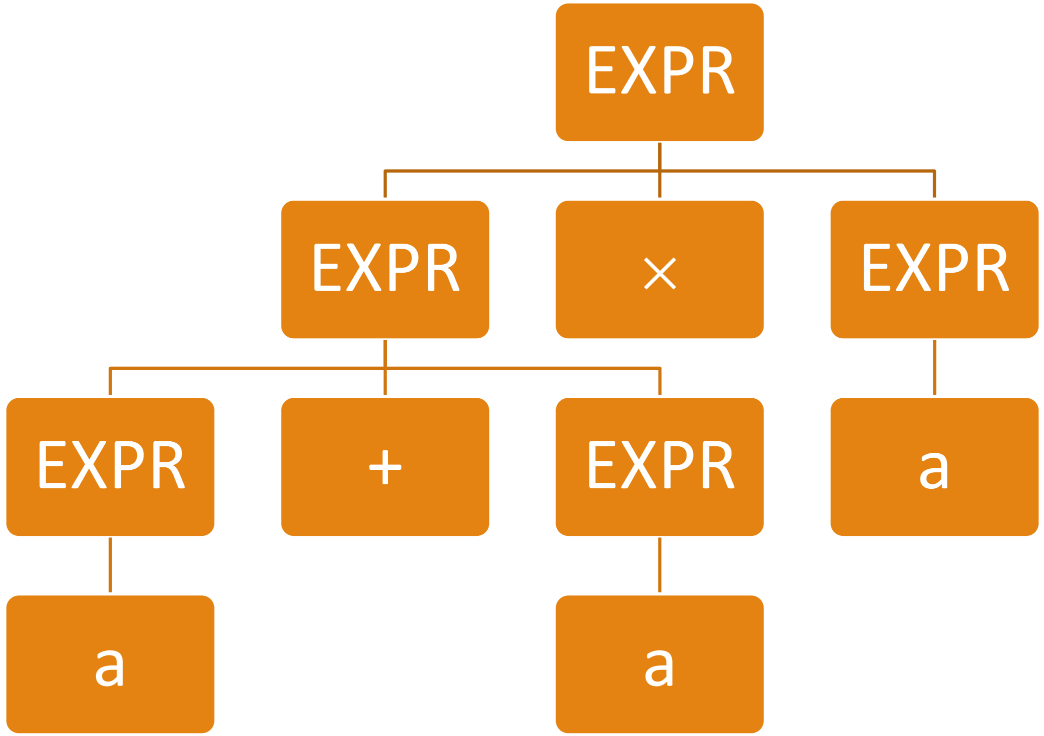
Rules *R* are…

EXPR → EXPR + EXPR

EXPR → EXPR × EXPR

EXPR → (EXPR)

EXPR → a



Ambiguity

V = { EXPR }

Σ = { **a**, +, ×, (,) }

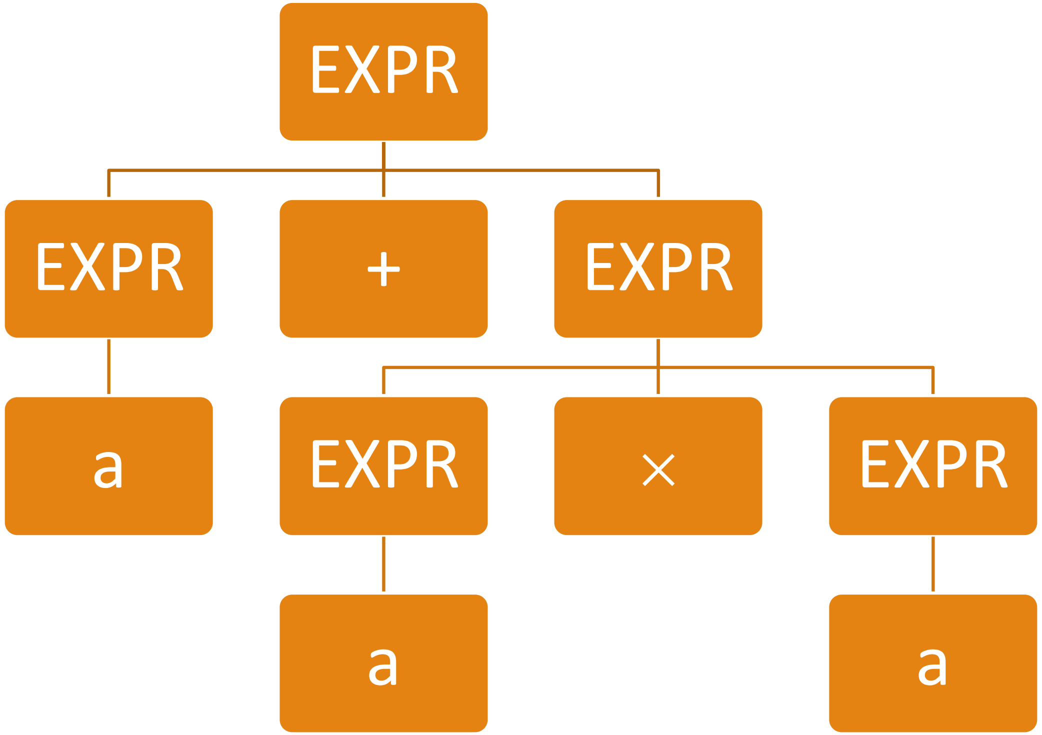
Rules R are...

EXPR → EXPR + EXPR

EXPR → EXPR × EXPR

EXPR → (EXPR)

EXPR → a



Ambiguity

A grammar may generate a string **ambiguously** by having two different parse trees for it

- That's *not* the same thing has having two different *derivations* – derivations can differ just by what order the rules are applied in
- To formalize ambiguity, we first define a **leftmost derivation** as a derivation in which, at every step, the leftmost remaining variable is the one replaced; that gives us:

Definition (Ambiguity):

- A string *w* is derived **ambiguously** in grammar ***G*** if it has two or more different leftmost derivations.
- A grammar ***G*** is **ambiguous** if it generates some string ambiguously.

Sometimes we can find an unambiguous grammar to generate the same language…

- …sometimes we can't
- There are languages that are **inherently ambiguous**

Chomsky Normal Form

A simplified form for context-free grammars

- Useful for working with CFGs using algorithms

A CFG is in **Chomsky Normal Form** if:

- Every rule is of one of the following forms:
  - $A \rightarrow BC$
  - $A \rightarrow \mathbf{a}$
  - $S \rightarrow \varepsilon$
- Where *A*, *B* and *C* are variables, **a** is a terminal, and:
  - *S* is the starting variable
  - Neither *B* nor *C* are *S* (*A* can be)

Converting to CNF: 4-Step Process

1. **Add a new start variable.**
   It rewrites only to the old start variable:
   - $S_0 \rightarrow S$
2. **Get rid of rewrites to the empty string.**
   For every rewrite of variable $X \rightarrow \varepsilon$:
   - Remove the rule $X \rightarrow \varepsilon$
   - Find every instance of a variable Y being rewritten to anything involving X
   - Add a new rule rewriting Y to the same thing, but with X removed
3. **Get rid of unit rules.**
   For every rewrite of variables $X \rightarrow Y$:
   - Remove the rule $X \rightarrow Y$
   - Find every instance of Y being rewritten to anything
   - Add a new rule rewriting X to the same thing
4. **Convert all the remaining rules.**
   For every rule $X \rightarrow y_1y_2y_3\ldots y_n$:
   - Remove the rule $X \rightarrow y_1y_2y_3\ldots y_n$
   - Make new rules $X \rightarrow y_1X_1$, $X_1 \rightarrow y_2X_2$, $X_2 \rightarrow y_3X_3$, …, $X_{n-2} \rightarrow y_{n-1}y_n$
     - When making these rules, for every y_i that's a terminal:
       - Replace it with a new variable Y_i
       - Create new rule $Y_i \rightarrow y_i$

CNF Conversion Example

(Board work: 2.10)

Next Time:
Pushdown Automata