## CSE396 Lecture Tue. 3/30: The Nature of CFIs and Non-CFLs

Prelim II exam probably **Tue. 4/20**.
Most common errors on Prelim I:

1. Not showing the dead state, nor arcs going to it, nor comments on why.
2. Confusing the strings $u, v \in \Sigma^*$ with individual characters while reading the condition $L = \{ubv : \#a(u) \geq \#a(v)\}$. Also some "too many stars" errors like $S = a^*b^*$.
3. Missing the $\epsilon$-arc $(2, \epsilon, 3)$ and how it makes the set-states $\{1, 2\}$ and $\{2\}$ impossible.

First, some more examples of structural induction (SI) on grammars: Consider $G =$

$$S \to aSa \mid bY \mid Yb,$$
$$Y \to YS \mid a \mid b.$$

And define $A = \left\{x \in \{a, b\}^* : |x| \textit{ is a positive even number}\right\}$. (Ignore the comma and the period---they are just punctuation, not part of $\Sigma = \{a, b\}$.)

The designer of $G$ intended $L(G) = A$, but missed an infinite number of strings. Let's first ask:

Is $G$ **sound**, i.e., is $L(G) \subseteq A$? Answer is yes. To prove it via SI, define the following:

- $P_S$ = "Every $x$ that I derive has $|x| \geq 2$ and $|x|$ even."
- $P_Y$ = "Every $y$ that I derive has $|y|$ odd."

The base cases are actually the latter two rules for $Y$, but in SI it does not matter what order you do things in. So start with the first-listed rule.

- $S \to aSa$: Suppose $S \Longrightarrow^* x$ using this rule first (utrf.). Then $x = aza$ where $S \Longrightarrow^* z$. By IH $P_S$ on the right-hand side (RHS), $|z|$ is a positive even number. Thus $|x|$, which equals $|z| + 2$, is also a positive even number. This upholds $P_S$ on the left-hand side (LHS).
- $S \to bY$: Suppose $S \Longrightarrow^* x$ utrf. Then $x = by$ where $Y \Longrightarrow^* y$. By IH $P_Y$ on RHS, $|y|$ is odd. The odd number can't be negative, so it must be $\geq 1$, and this makes $|x| = |y| + 1$ an even number $\geq 2$. That upholds $P_S$ on the LHS.
- $S \to Yb$: "Similar to the previous case." [We are done with $S$ but that's not enough.]
- $Y \to YS$: Suppose $Y \Longrightarrow^* y$ utrf. Then $y = uv$ where $Y \Longrightarrow^* u$ and $S \Longrightarrow^* v$. By IH $P_Y$ on RHS, $|u|$ is odd, and by IH $P_S$ on RHS, $|v|$ is even. Hence $|y|$ equals an odd number plus an even number, which is odd---and upholds $P_Y$ on LHS.
- $Y \to a$: Since $|a| = 1$ which is odd, $P_Y$ is immediately satisfied.
- $Y \to b$: Similar.

$$S \to aSa \mid bY \mid Yb,$$
$$Y \to YS \mid a \mid b.$$

Thus $P_S$ and $P_Y$ are upheld in all rules. Thus every $x$ such that $S \implies^* x$ has positive even length, which means it belongs to $A$. So $L(G) \subseteq A$. ⊠

This makes $G$ sound, but is it **comprehensive**, meaning $A \subseteq L(G)$, i.e. $L(G) \supseteq A$? Here again are the definitions: $A = \{x \in \{a,b\}^* : |x| \textit{ is a positive even number}\}$ and $G =$

$$S \to aSa \mid bY \mid Yb,$$
$$Y \to YS \mid a \mid b.$$

The shortest strings in $A$ have length 2. Can we get all four of them? We have $S \implies bY \implies ba$ and $S \implies bY \implies bb$, then $S \implies Yb \implies ab$ and $S \implies Yb \implies bb$ again---oh look, ambiguity! But we cannot get $aa$. Can we get $aaaa$? any even-length string with only $a$'s?

No, we cannot, so $G$ misses being comprehensive "by an infinite mile." Is there an easy way to prove that $A \setminus L(G)$ is in fact infinite---because it contains $(aa)^+$? SI can also be used to prove "uncomprehensiveness". The deft way is to do "induction loading" to augment $P_S$ to:

- $P'_S = P_S \wedge$ "Every $x$ that I derive has at least one $b$."

The rules $S \to bY$ and $S \to Yb$ uphold this immediately, so we need only consider $S \to aSa$: Suppose $S \implies^* x$ utrf. Then $x = aza$ where $S \implies^* z$. By IH $P'_S$ on the right-hand side (RHS), $|z|$ is a positive even number and $z$ has at least one $b$. Thus $|x|$, which equals $|z| + 2$, is also a positive even number, and $x$ has the at-least-one-$b$ from $z$. This upholds $P'_S$ on LHS. So $L(G) \subseteq A \setminus a^*$ (which incidentally equals $A \setminus (aa)^+$).

Is $L(G) = A \setminus (aa)^+$ ? A good self-study question...though we are not fully covering comprehensiveness proofs by induction on (the lengths of) strings. A related question: if we add the rule $S \to aa$, does that make the resulting grammar $G'$ accept $A$?

A second example illustrates how SI accompanies reasoning about programming syntax. The following is a viable grammar for a fragment of Java:

```
E   -->  E2 ASSGTOP E | E2          //assignment is right-associative
E2  -->  E2 BINOP E3 | E3           //BINOPs are left-associative
E3  -->  +E3 | -E3 | ++P | --P | E4
E4  -->  P++ | P-- | P
P   -->  (E) | LITERAL | VARIABLE   (etc.)
ASSGTOP --> = | += | -=   (etc.)
BINOP   --> == | != | + | - | * | /   (etc.)
LITERAL --> /any number or quoted string etc./
VARIABLE--> /any legal identifier/.
```

1. Show how to derive a legal Java expression that has the substring "`++ + ++`" in it, noting the whitespace around the binary +.
2. Prove by ``structural induction'' that the substring "+ ++ +" can never occur in a legal Java expression, nor any similar one with - in place of + and/or -- in place of ++.

For (a), there is the derivation:

```
E ==> E2 ==> E2 BINOP E3 ==> E3 BINOP E3
  ==> E4 BINOP E3 ==> P++ BINOP E3 ==>^2 x++ BINOP E3 ==> x++ + E3
  ==> x++ + ++P ==>^2 x++ + ++y
```

Assign to the grammar nonterminal `P` the meaning, "Every string $x$ I derive begins and ends with a parenthesis, alphanumeric char, or quotes." This is immediately preserved by the productions for `P`. Then we can assign to all of `E,E2,E3,E4,P` the meaning,

"Any ++ or -- token in a string I derive is immediately preceded or followed by a parenthesis, alphanumeric char, or quotes."

Note that for `P` this adds to the previous meaning--it is needed to ``inherit'' this to `P` because `P` can derive `E`. Then the rules preserve this latter meaning too---immediately in the case of right-hand sides ++P, --P, P++, P-- because of the meaning of `P`, and by induction for the other right-hand sides. The most sensitive cases are where E3 is preceded by a unary + or - or BINOP on the right-hand side. Then the point is that any ++ or -- derived by the E3 on the right-and side (and by the E2 preceding the BINOP) is going to have a non-operator precede or follow it. We don't have to worry about the words "it isn't preceded by an operator" possibly allowing +E3 to violate because we "accentuated the positive"---we said any ++ or -- "in" E3 is *positively either preceded or followed by a non-operator*, which takes care of anything the + in +E3 could threaten. Ditto with either side of BINOP or the front side of ASSGTOP.

```
E   -->  E2 ASSGTOP E | E2         //assignment is right-associative
E2  -->  E2 BINOP E3 | E3          //BINOPs are left-associative
E3  -->  +E3 | -E3 | ++P | --P | E4
E4  -->  P++ | P-- | P
P   -->  (E) | LITERAL | VARIABLE   (etc.)
ASSGTOP --> = | += | -=   (etc.)
BINOP   --> == | != | + | - | * | /   (etc.)
```

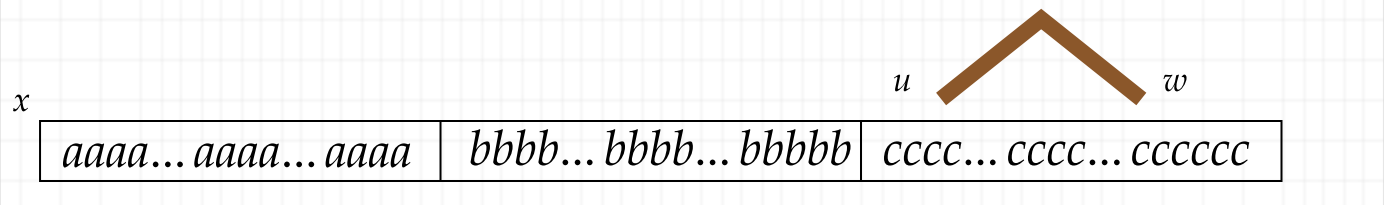
Now we will consider how the CFL Pumping Lemma impacts programming syntax and compiler design. Before getting to that fully, let us revisit the example from the end of last time: Prove that

$$L = \left\{a^i b^j c^k : i < j \wedge j < k\right\}$$

is not a CFL. We will follow the "adversary" version of the "proof script":

Adv: "I have a CFG $G$ such that $L(G) = L$"?
You: What is $N = 2^{|V'|}$ when you convert your $G$ into a CFG $G'$ in Chomsky normal form?
Adv: "$N$".
You take $x = a^N b^{N+1} c^{N+2}$ and say: "Give me a breakdown of $x$ into $yuvwz$ such that $|uvw| \leq N$ and at least one of $u$ and $w$ is not the empty string."

You have to be prepared for every kind of case the adversary could come up with




Claim: The following two cases are *exhaustive*: Either

- $u$ or $w$ includes at least one $a$ or at least one $b$ but does not include any $c$'s, or
- $u$ or $w$ includes at least one $b$ or at least one $c$ but does not include any $a$'s, or

These cases are exhaustive because the "compass" cannot be wider than the $b^{N+1}$ part.

In the first case, we "pump up". If $uw$ *includes at least one* $b$ then pumping up creates at least $N+2$ $b$'s while the number of $c$'s remains at $N+2$, so the $k > j$ condition is violated. Else, $uw$ includes only $a$'s, but then pumping up violates the $j > i$ condition.

In the second case, we "pump down": Either this subtracts at least one $b$, thus violating $i < j$ since $i = N$ is left alone in the $a$'s, or this subtracts at least one $c$ without changing the number of $b's$, thus violating $j < k$.

The rule of thumb is that a CFG can handle one kind of counting dependency but not two such dependencies that "entangle" in some way.

Now consider the following three languages:

1. $L_1 = \left\{a^m b^m a^n b^n : m, n \geq 1\right\}$,
2. $L_2 = \left\{a^m b^n a^m b^n : m, n \geq 1\right\}$,
3. $L_3 = \left\{a^m b^n a^n b^m : m, n \geq 1\right\}$

Which one(s) are CFLs and which not?