CSE396 Lecture Thu. Week 8: More Proofs With CFGs

The most prominent uses of Structural Induction---one can say explicitly, not just implicitly---is in the back-end steps of a compiler *after* **parsing** the given code. The theorem involved is that the conversion from parsed program code to (abstract syntax trees and then into) object code is uniquely defined and mathematically, logically correct. Here (FYI) is an accessible reference by Ruslan Spivak (2015) for building a simple interpreter of mostly arithmetical expressions.

Part of this is making sure that incompatible operations do not get juxtaposed. For one example, while it is legal in the C/C++ family of languages to do repeated pre-increment:

```
int y = ++(++x);
```

repeated *post*-increment `y = (x++)++;` is illegal. The "semantic" reason is that post-increment `x++` returns a *pure value* that is not tied to a memory location, whereas pre-increment `++x` returns a *reference* to the *storage object* x. The BNF grammar treats pre-increment and post-increment in different syntactic categories---i.e., using different grammar variables and rules---so that only the former can be juxtaposed to itself (with-or-without parentheses). [I spent a fair bit of time demo'ing this in the file https://cse.buffalo.edu/~regan/cse396/AK/CSE396parses.c --- I was hoping to catch more people doing the current CSE305 project.]

We can get an abstract taste of this from a third kind of "SI" example: forbidden substrings. Here is the example at https://cse.buffalo.edu/~regan/cse396/CSE396lect040518.pdf : $G =$

$$\begin{aligned} S &\rightarrow \epsilon \mid b \mid AS \mid SC \\ A &\rightarrow a \mid bCaA \\ C &\rightarrow aS \mid CC \end{aligned}$$

The target property is $T = \{x \in \{a,b\}^* : \ x \textit{ does not have } bb \textit{ as a substring}\}$.

Prove $L(G) \subseteq T$.

We know we want P_S to include "Every x that I derive does not have bb as a substring." Is this strong enough to hold up recursively? Well, we need to define properties P_A and P_C first:

P_A = "Every y that I derive _____ ..."
P_C = "Every z that I derive _____ ..."

[The first student attempt at filling in the blanks was simply to repeat P_S by saying that strings derived from A and C do not have a bb substring either. But, I pointed out, what if a string y derived from A ends in a single b? Then the rule $S \rightarrow AS$, together with $S \rightarrow b$, can give us yb, which is not allowed as it ends in bb. So we realized that we need these properties:]

P_A = "Every y that I derive ___ends in an a____ ."
P_C = "Every z that I derive ____starts with an a_____ ."

[Then, after saying it doesn't matter what order you do the rules in, I took it from the end:]

$C \to aS$: no induction needed, immediate
$C \to CC$: Suppose $C \Longrightarrow^* z$ utrf. Then $z =: uv$ where $C \Longrightarrow^* u$ and $C \Longrightarrow^* v$. By IH P_C on the RHS, u begins with a. Hence z begins with a, and this upholds P_C on the LHS.

$A \to a$: Immediate, since the string "a" ends in a (and does not have bb inside it).
$A \to bCaA$: Self-recursive on the end, like $C \to CC$ was on the beginning. Is there anything more we should say?

[Well...after moving on to the rules of S we realized that although P_A and P_C are what we need for ruling out bb being produced at the *junctures* between two substrings, we haven't actually ensured that the variables A and C on the whole don't produce a bb internally. So there is more we should say! We blended in the first student suggestion after all, strengthening the properties to read:]

P_A = "Every y that I derive ___ends in an a____ && P_S (i.e., y does not have a bb substring)."
P_C = "Every z that I derive ____starts with an a_____ && P_S."

The coverage of rules then has to be upgraded too, upholding the "no internal bb" clauses as well"

<u>Rules</u>
$C \to aS$: no induction needed, immediate---well, we need to attend to the S part too. So suppose $C \Longrightarrow^* z$ using this rule first. Then $z = aw$ where $S \Longrightarrow^* w$. By IH P_S on RHS, w has no internal bb. It cannot make a bb with the leading a either. So the "&& P_S" part holds for z too.

$C \to CC$: Suppose $C \Longrightarrow^* z$ utrf. Then $z =: uv$ where $C \Longrightarrow^* u$ and $C \Longrightarrow^* v$. By IH P_C on the RHS, u begins with a. Hence z begins with a, and this upholds P_C on the LHS. Moreover, u and v themselves do not have bb as a substring, **and** even if u ends in a b, the string v must start with an a by IH P_C for the second C on the RHS, so the juncture between u and v does not have bb.

$A \to a$: Immediate, since the string "a" ends in a (and does not have bb inside it).

$A \to bCaA$: Self-recursive on the end, like $C \to CC$ was on the beginning. Is there anything more we should say? **Yes**: Suppose $A \Longrightarrow^* y$ using this rule first. Then $y =: buav$ where $C \Longrightarrow^* u$ and $A \Longrightarrow^* v$, By IH P_C on the RHS, u begins with a---so the juncture does not cause a bb---and u has no internal bb. The juncture of u on the RHS is OK since the next char is an a, and also hence, v can't cause bb at that juncture, nor internally by IH P_A. This now upholds **all** of the property P_A on the LHS.

[In lecture I did the rules of S orally. Here they are in full:]

$S \to \epsilon$ and $S \to b$: Immediate. The latter causes "danger" by introducing a b, but does not violate P_S` since it doesn't have bb.

$S \to AS$: Suppose $S \Longrightarrow^* x$ utrf. Then $x = yz$ where $A \Longrightarrow^* y$ and $S \Longrightarrow^* z$. By IH P_A and P_S on the RHS, y and z do not have any internal bb, and y ends with a. This rules out a bb anywhere in x, so P_S on LHS is upheld.

$S \to SC$: Suppose $S \Longrightarrow^* x$ utrf. Then $x = yz$ where $S \Longrightarrow^* y$ and $C \Longrightarrow^* z$. By IH P_S and P_C on the RHS, y and z do not have any internal bb, and z begins with a. This rules out a bb anywhere in x, so P_S on LHS is again upheld. This completes the whole proof of $L(G) \subseteq T$ by SI. ☒

Proofs of Comprehensiveness, $L(G) \supseteq T$:

These work by induction on the **lengths** n of strings. A variable A gets an induction property R_A of the form:

R_A = "For all n, and for each y of length n, if y satisfies _____ then $A \Longrightarrow^* y$."

We can also write this as $\forall n R_A(n)$, where $R_A(n)$ is the part beginning "for each y of length n." The base case is $n = 0$; sometimes we augment the basis by showing $n = 1$ and/or $n = 2$ directly. Then we use the **strong induction hypothesis** "For all $m < n$, $R_A(m)$."

If the grammar has variables S, A, B, C, say, then we need to craft properties R_S, R_A, R_B, and R_C. The global property we need to prove is:

$$\forall n P(n), \text{ where } P(n) = R_S(n) \wedge R_A(n) \wedge R_B(n) \wedge R_C(n).$$

Thus we can get thrown into a multithreaded mutual recursion here too. Our first example will just use a property of S though.

Example 1: $G = S \to SS \mid (S) \mid \epsilon,\ T = BAL.$

$R_S(n) \equiv$ for each $x \in \{(,)\}^n$, if x is balanced then $S \Longrightarrow^* x$. [The symbol $\equiv$ can be read as "expresses" or "is defined as."] We need to prove $\forall n R(n)$. To define "x is balanced," we mandated that $diff(x, i) \geq 0$ for all i, where $diff(x, i)$ means the number of '(' chars in the first i bits of x, minus the number of ')' chars in those bits, and also $diff(x, n) = 0$, where $n = |x|$, the length of x. We prove this using "strong induction", aka. "course-of-values induction.":

Basis ($n = 0$): The only string in Σ^0 is ϵ. We count ϵ as balanced, so the premise of $R_S(0)$ is not vacuous. But the conclusion holds by our having the rule $S \to \epsilon$, so $S \Longrightarrow^* \epsilon$, so $R_S(0)$ holds.

Induction ($n \geq 1$): we may assume the **strong induction hypothesis** "For all $m < n$, $R_S(m)$." Goal: prove $R_S(n)$. Here a student question led to the important point that the case $n = 1$ does hold "vacuously"---i.e., by default---since no strings of length 1 (or any odd length) are balanced. We could have included $n = 1$ as a second base case, but formally it is fine as treated here.

[Rest done on chalkboard at https://cse.buffalo.edu/~regan/cse396/CSE396S26Week8ThuCB.jpg]

Since we previously proved $L(G) \subseteq BAL$ by SI, this gives us finally $L(G) = BAL$, so this grammar is **correct**.

Example 2: $G = S \to \epsilon \mid aB \mid bA,\ A \to aS \mid bAA,\ B \to bS \mid aBB;\ T = \{x : \#a(x) = \#b(x)\}$.

This needs defining auxiliary languages and properties:

- $L_A = \left\{x \in \{a,b\}^* : \#a(x) = \#b(x) + 1\right\}$.
- $L_B = \left\{x \in \{a,b\}^* : \#b(x) = \#a(x) + 1\right\}$.

- $R_A(n) \equiv$ for each x of length n, if $\#a(x) = \#b(x) + 1$ then $A \Longrightarrow^* x$.
- $R_B(n) \equiv$ for each x of length n, if $\#b(x) = \#a(x) + 1$ then $B \Longrightarrow^* x$.

We have to prove these as well as $R_S(n) \equiv$ for each x of length n, if $\#a(x) = \#b(x)$ then $S \Longrightarrow^* x$. That is, we have to prove the global property $\forall n P(n)$ where $P(n) \equiv R_S(n) \wedge R_A(n) \wedge R_B(n)$. As with SI, this again requires going through each grammar rule, but with the goal of **building** a derivation in the first place, rather than combining properties of derived (sub-)strings. The base case $n = 0$ is handled by $S \Longrightarrow \epsilon$ again, so we need the induction case $n \geq 1$. The value $n = 1$ is not vacuous for the component properties $R_A(n)$ and $R_B(n)$ this time, but we can still include it in the induction.

I showed one rule to handle one case of showing $R_A(n)$: Let any x of length n be given such that $\#a(x) = \#b(x) + 1$. Since $n \geq 1$, we can break into (sub-)cases (i) x begins with a and (ii) x begins with b. In case (i), $x = ay$ where $\#a(y) = \#b(y)$. Since $|y| = n - 1 < n$, we can use the strong IH $R_S(n-1)$ to conclude $S \Longrightarrow^* y$. Then we build the derivation $A \Longrightarrow aS \Longrightarrow^* ay = x$. So we get $A \Longrightarrow^* x$, which is what we needed to show. Since x was an arbitrary string in L_A of length n, we get $R_A(n)$ in subcase (i), but we still need to do subcase (ii).

In subcase (ii), $x = bw$ where w must have **two** more a's than b's. By what can be analogized to the Intermediate Value Theorem of calculus, it must be possible to break w into strings u and v that each

have exaclty one more a than b. This was diagrammed in the photo

https://cse.buffalo.edu/~regan/cse396/CSE396S26Week8ThuCB2.jpg

Individually, $m_1 = |u|$ and $m_2 = |v\}$ must both be $< n$. So we can use the strong induction hypotheses $R_A(m_1)$ and $R_A(m_2)$ to obtain the derivations $A \Longrightarrow^* u$ and $A \Longrightarrow^* v$. Then we finally build the derivation

$$A \Longrightarrow bAA \Longrightarrow^* buA \Longrightarrow buv = bw = x.$$

Since we've handled both subcases (i) and (ii), this shows $R_A(n)$. Showing the other components $R_B(n)$ and $R_S(n)$ finally shows $P(n)$, and $\forall n P(n)$ giving $L(G) \supseteq T$ follows by induction. $\boxtimes$

Since we proved $L(G) \subseteq T$ by SI in the previous lecture, this finally---rigorously---proves the claim $L(G) = T$.