Software Testing, Quality Assurance and Maintenance	Winter 2025
Lecture 5b — January 22, 2025	
Patrick Lam	*version 1*

I said I'd do a structural induction example in class since there isn't one in the notes yet.

For a structure built from a grammar, we would use the grammar to provide the proof obligations for the structural induction. WHILE has terminals, arithmetic expressions, Boolean expressions, and statements.

Proposition 1 *All WHILE programs that do not contain any* `while` *statements always terminate.*

Proof. We are going to use the big-step semantics to prove this. In particular, we are going to show that any `while`-less program is going to admit a finite derivation that finishes with some final state.

A WHILE program is an `slist`, which is one or more statements. Statements may contain arithmetic or boolean expressions.

Lemma 1 *Evaluation of any boolean or arithmetic expression always yields a value and terminates.*

Proof of lemma. We first consider arithmetic expressions. The base case for these expressions are integers n and variables x. The semantics contains rules:

$$\frac{}{\langle n, q\rangle \Downarrow n} \quad \text{and} \quad \frac{}{\langle x, q\rangle \Downarrow q(x)} \cdot$$

which clearly yield values and terminate. Next, we have the three grammar rules that build arithmetic expressions from smaller ones: negation, parentheses, and arithmetic operators. For each of those, we assume that all smaller expressions yield values and terminate. I'll show the proof for the arithmetic operator $+$. (I went and sneakily removed division...).

$$\frac{\langle e_1, q\rangle \Downarrow n_1 \qquad \langle e_2, q\rangle \Downarrow n_2}{\langle e_1 + e_2, q\rangle \Downarrow n_1 + n_2}.$$

We assume, by induction, that e_1 and e_2 have the desired property. This derivation tree shows that if you build an arithmetic expression with $+$, then it also has the desired property. It would be OK to say "similarly for - and *". You should probably show at least one unary operator case too.

$$\frac{\langle e, q\rangle \Downarrow n}{\langle (e), q\rangle \Downarrow n}.$$

Here we say that because e has the property, this derivation shows that (e) also yields a value and terminates. Unary negation is the same.

For boolean expressions, the base cases are `true` and `false`, and you can quote the inference rules

$$\frac{}{\langle \texttt{true}, q\rangle \Downarrow \texttt{true}} \quad \text{and} \quad \frac{}{\langle \texttt{false}, q\rangle \Downarrow \texttt{false}} \cdot$$

The relational operators rely on us having shown termination for arithmetic expressions.

$$\frac{\langle e_1, q\rangle \Downarrow n_1 \qquad \langle e_2, q\rangle \Downarrow n_2}{\langle e_1 < e_2, q\rangle \Downarrow (n_1 < n_2)}.$$

That is, we know that e_1 and e_2 evaluate to integers, and then we apply the rule and return `true` if $n_1 < n_2$ and `false` otherwise. Finally, the boolean operators rely on the induction hypothesis.

$$\frac{\langle e_1, q\rangle \Downarrow t_1 \qquad \langle e_2, q\rangle \Downarrow t_2}{\langle t_1 \ \texttt{and} \ t_2, q\rangle \Downarrow (t_1 \wedge t_2)}.$$

We assume that e_1 and e_2 evaluate to boolean values t_1 and t_2 and then we have a rule which yields a value for t_1 and t_2. Should mention unary operator `not` and parenthesized expressions also. ∎

Returning to the proof for statements, then, we have a number of statements for which we need to prove the claim. The base cases for this proof are `skip`, assignment statements, and the `print_state`, `assert`, `assume`, and `havoc` statements. We only gave semantics for `skip` and assignment, so we'll not talk about the other statements either.

$$\frac{\langle e, q\rangle \Downarrow n}{\langle x := e, q\rangle \Downarrow q[x := n]}.$$

This is a base case for this proof because there is no statement in the hypothesis. We rely on the lemma to show that we can evaluate e to value n and terminate. This rule shows that we can evaluate an assignment statement and terminate. (No more yielding a value here in the big-step semantics; evaluation just changes the state.)

The inductive cases are the statement list `slist` and the if statement. I guess I can show them both. For if, we have termination because we inductively assume termination for the then and else clauses.

$$\frac{\langle s_1, q\rangle \Downarrow q' \qquad \langle e, q\rangle \Downarrow \text{true}}{\langle \texttt{if} \ e \ \texttt{then} \ s_1 \ \texttt{else} \ s_2, q\rangle \Downarrow q'}.$$

This is the then clause. You can say "similarly for `else`". We have s_1 terminates by induction and so the `if` also terminates by this derivation.

Finally, for the statement list:

$$\frac{\langle s_1, q\rangle \Downarrow q^1 \qquad \cdots \qquad \langle s_n, q^{n-1}\rangle \Downarrow q^n}{\langle \{s_1; \cdot; ss_n\}, q\rangle \Downarrow q^n}.$$

Because, inductively, all of the s_i terminate, then so does the list of s. ∎