22C:185 - Fall 2004

Consistency of the Denotational and Axiomatic Semantics of Wren

Theorem: The denotational and axiomatic semantics of Wren with arrays are consistent. That is, for any Wren command *C* and assertions P and Q, if ⊢ {P} *C* {Q}, and sto is a store so that evaluate⟦P⟧ sto = true, then evaluate⟦Q⟧ sto' = true where execute⟦*C*⟧ sto = sto'.

Proof: (by structural induction)

I. We first show that the theorem holds for each of the atomic statements.

A. **skip**: *C* = **skip**

This case is clear by inspection.

B. Assignment — simple variable: *C* = X:= *e*

In this case P = Q[X→ *e*], and evaluate⟦Q[X→ *e*]⟧ sto will proceed identically to evaluate⟦Q⟧ sto' except where X is encountered in Q. Where X is encountered in Q, *e* appears in P, and since sto' = execute⟦*C*⟧ sto = updateSto(sto, X, evaluate⟦*e*⟧ sto), evaluate⟦X⟧ sto' = evaluate⟦*e*⟧ sto . Since at every other point the evaluation steps are identical, evaluate⟦Q⟧ sto' = evaluate⟦P⟧ sto = true. That is, evaluating X in store sto' yields the same result as evaluating *e* in state sto, and since there are no other points of difference if P evaluates true in sto, Q must evaluate true in sto' . A rigorous proof of this case is actually another induction based on the structure of Q.

C. Assignment — subscripted variable: *C* = A[I]:= *e*

We assume that in the denotational semantics for subscripted variables, all the functions associated with stores accept subscripted variables (e.g., A[1], A[2], etc.) as well as simple variables. Then the 'evaluate' function is extended with the additional case:

evaluate⟦A[I]⟧ sto = applySto(sto, A[v]), where v = evaluate⟦I⟧ sto

Also, to be applicable to pre/post-conditions, an extension to evaluate the array value notation is necessary, namely:

evaluate⟦A<I:*e*>[j]⟧ sto = **if** evaluate⟦I⟧ sto = evaluate⟦j⟧ sto
then evaluate⟦*e*⟧ sto **else** evaluate⟦A[j]⟧ sto

Then the added semantic equation for subscripted variable assignment is:

execute ⟦A[I]:= *e*⟧ sto = updateSto(sto, A[v], (evaluate⟦*e*⟧ sto)),
where v = evaluate⟦I⟧ sto.

Now for the proof. In this case P = Q[A→ A<I:*e*>], and evaluate⟦Q[A→ A<I:*e*>]⟧ sto will proceed identically to evaluate⟦Q⟧ sto' except where A is encountered in Q. Where A is encountered in Q, A<I:*e*> appears in P. Hence an evaluation of A[E] in Q using store sto' is replaced by an evaluation of A<I:*e*>[E[A→ A<I:*e*>]] using store sto in P. The potential difference in the subscripts requires careful analysis. In fact, the proof in this case employs a (sub) induction on the depth of nesting of subscripted references to A. For instance, A[2*A[A[3]+1]] has nesting level 2.

Basis case: n=0 (no nesting)

If there is no nesting, then A[E] in Q is replaced by P by A<I:*e*>[E] in P. Since E does not involve A, its evaluation in sto and sto' produces exactly the same result

(only A has been changed), say evaluate⟦E⟧ sto = evaluate⟦E⟧ sto' = p. But then evaluate⟦A<I:*e*>[p]⟧ sto = evaluate⟦A[p]⟧ sto'. Since there are no other points of difference, if P evaluates true in sto, Q must evaluate true in sto'. A rigorous proof of this case is actually another induction based on the structure of Q.
<u>Induction step</u>: n=n+1
The induction hypothesis is that for all subscripted variable references A[E] with depth of nesting n, evaluate⟦A<I:*e*>[E[A□ A<I:*e*>]]⟧ sto = evaluate⟦A[E]⟧ sto'. Let A[F] be a subscripted variable reference in Q with depth of nesting n+1. Then at the corresponding position in P we find A<I:*e*>[F[A□ A<I:*e*>]]. But F has depth of nesting n and hence evaluate⟦F[A□ A<I:*e*>]⟧ sto = evaluate⟦F⟧ sto' = p. Then

```
evaluate⟦A<I:e>[F[A□ A<I:e>]]⟧ sto = if evaluate⟦I⟧ sto = p
                                     then evaluate⟦e⟧ sto
                                     else evaluate⟦A[F]⟧ sto
= evaluate⟦A[p]⟧ sto' = evaluate⟦A[F]⟧ sto'.
```

II. We next show that the theorem must be true for any compound statement, assuming that it is true for its constituent statements.

A. Sequential control: $C = C_1; C_2$

If |□ {P} C {Q}, then for some assertion R, |□ {P} C_1 {R} *and* |□ {R} C_2 {Q}**.** By the inductive hypothesis we assume that the result is true for the constituent commands C_1 and C_2. That is, for any sto with evaluate⟦P⟧ sto = true and execute⟦C_1⟧ sto = sto_1, then evaluate⟦R⟧ C_1 = true, and then if execute⟦C_2⟧ sto_1 = sto_2, evaluate⟦Q⟧ sto_2 = true. But execute⟦C_1; C_2⟧ sto = sto_2, and so this case is proven.

B. **if-then-else**: C = **if** B **then** C_1 **else** C_2 **end if**

If |□ {P} **if** B **then** C_1 **else** C_2 {Q}, then |□ {P □ B} C_1 {Q} and |□ {P □ ¬B} C_2 {Q}**.** By the inductive hypothesis we assume that the result is true for the constituent commands C_1 and C_2. That is, if for sto, evaluate⟦P □ B⟧ sto = true and execute⟦C_1⟧ sto = sto_1, then evaluate⟦Q⟧ sto_1 = true, and if evaluate⟦P □ ¬B⟧ sto = true and execute⟦C_2⟧ sto = sto_2, then evaluate⟦Q⟧ sto_2 = true. But if evaluate⟦P □ B⟧ sto = true, then evaluate⟦B⟧ sto = true in which case execute⟦**if** B **then** C_1 **else** C_2⟧ sto = sto_1, and so this case is proven. Similarly if evaluate⟦P □ ¬B⟧ sto = true, then evaluate⟦¬B⟧ sto = true and hence evaluate⟦B⟧ sto = false. But then execute⟦**if** B **then** C_1 **else** C_2⟧ sto = sto_2, and so this case is also proven.

C. **If-then**: C = **If** B **then** C_1 **end if**

Similar to case II.B above.

D. **while-do**: C = **while** B **do** C_1

Finally, suppose |□ {P} **while** B **do** C_1 {P □ ¬B}**.** Then |□ {P □ B} C_1 {P} and by the inductive hypothesis we assume that the result is true for the constituent command C_1. Now assume that sto is a store with evaluate⟦P⟧ sto = true and execute⟦**while** B **do** c_1⟧ sto = sto' (if it exists). To prove this case, we need to prove that the post-condition, evaluate⟦P □ ¬B⟧ sto' = true. To prove this we need to examine the definition of the state sto' . This is defined recursively,

□ sto, if evaluate⟦B⟧ sto = false
sto' =□
□execute⟦**while** B **do** C_1⟧ sto_1, if evaluate⟦B⟧ sto = true,

where sto_1 = execute⟦C_1⟧ sto . The truth of the post-condition is proven by a (sub) induction on the depth of this recursion.

<u>(sub) Basis case</u>(depth 0 - no recursion): if there is no recursion, then evaluate⟦P⟧ sto = true, evaluate⟦B⟧ sto = false, and sto = sto' , so evaluate⟦P □ ¬B⟧ sto' = evaluate⟦P □ ¬B⟧ sto = evaluate⟦P⟧ sto □ evaluate⟦¬B⟧ sto = true.

<u>(sub) Induction step</u>: assume that the post-condition is true for all recursions of depth n≥0, and let sto' result from a recursion of depth n+1. Then evaluate⟦B⟧ sto = true and sto' = execute⟦**while** B **do** C_1⟧ sto_1, where sto_1 = execute⟦C_1⟧ sto . But then execute⟦**while** B **do** C_1⟧ sto_1 must lead to a recursion of depth n. Now since evaluate⟦P □ B⟧ sto = true in this case, and |□ {P □ B} C_1 {P}, by the main induction hypothesis, evaluate⟦P⟧ sto_1 = true. Now if evaluate⟦B⟧ sto_1 = false, then sto' = sto_1 and evaluate⟦P □ ¬B⟧ sto' = evaluate⟦P □ ¬B⟧ sto_1 = evaluate⟦P⟧ sto_1 □ evaluate⟦¬B⟧ sto_1 = true. Or if evaluate⟦B⟧ sto_1 = true, then sto' results from a recursion of depth n and by the depth of recursion induction hypothesis, sto' satisfies the post-condition. Thus the (sub) induction is extended in either case, and the proof of the **while**-command is complete.

This completes the main induction step for each of the constituent command types and completes the proof for all commands.

III. Finally, we show that the theorem holds for strengthening pre-conditions, weakening post-conditions, and array-value deductions.

A. Strengthening the pre-condition

Suppose that |□ {P} *C* {Q} follows from |□ {R} *C* {Q} and |□ P R, where by induction, the theorem holds for |□ {R} *C* {Q}. Suppose sto is a store so that evaluate⟦P⟧ sto = true. But then since |□ P R, evaluate⟦R⟧ sto = true. But then by the inductive assumption, evaluate⟦Q⟧ sto' = true where execute⟦*C*⟧ sto = sto' .

B. Weakening the post-condition

Similar to case III.A above.

C. Evaluating array values

The definition of evaluate⟦A<I:*e*>[j]⟧ sto (see case I.C) makes the validity of the array-value proof rules immediately obvious.