Proof Outlines; Convergence

CS 536: Science of Programming, Fall 2022

Due Thu Nov 3, 11:59 pm

p.3

A. Why?

- A formal proof lets us write out in detail the reasons for believing that something is valid.
- Proof outlines condense the same information as a proof.

B. Outcomes

After this homework, you should be able to

- Translate between full proof outlines and formal proofs of partial correctness.
- Translate between a full proof outline and a minimal proof outline.

C. Problems [60 points total]

Classes 16 &17: Proof Outlines [35 points]

1. (Full outline from formal proof) Show the full outline derived from the full proof.

1.	{n > 0} k := n−1 {n > 0 ∧ k = n−1}	assignment (forward)
2.	{n > 0 ∧ k = n−1} x := n {n > 0 ∧ k = n−1 ∧ x = n}	assignment (forward)
3.	n > 0 ∧ k = n−1 ∧ x = n → p // where p ≡ 1 ≤ k ≤ n ∧ x = n!/k!	predicate logic
4.	{n > 0 ∧ k = n−1} x := n {p}	postcondition weak. 2, 3
5.	{n > 0} k := n−1 ; x := n {p}	sequence 1, 4
6.	{p[x*k/x]} x := x*k {p}	assignment (backward)
7.	{p[x*k/x][k−1/k]} k := k−1 {p[x*k/x]}	assignment (backward)
8.	p ∧ k > 1 → p[x*k/x][k−1/k]	predicate logic
9.	{p ∧ k > 1} k := k−1 {p[x*k/x]}	precondition strength. 8, 7
10.	{p ∧ k > 1} k := k−1 ; x := x*k {p}	sequence 9, 6
11.	{ **inv** p} W {p ∧ k ≤ 1} // where W ≡ **while** k > 1 **do** k := k−1 ; x := x*k **od**	while loop 10
12.	{n > 0} k := n−1 ; x := n { **inv** p} W {p ∧ k ≤ 1}	sequence 5, 11

Expanded substitutions: (You don't have to re-include this with your outline)

p ≡ 1 ≤ k ≤ n ∧ x = n!/k!

p[x*k/x] ≡ 1 ≤ k ≤ n ∧ x*k = n!/k!

p[x*k/x][k−1/k] ≡ 1 ≤ k-1 ≤ n ∧ x*(k-1) = n!/(k-1)!

2. [10 pts] Give a full proof outline obtained by expansion of the partial proof outline below. Work forward through the program (use sp on the four assignments and **if-else** statement). If you use p[e/v] substitution notation, show their resulting expansions somewhere.

    {q ≡ r = X*Y–x*y}
    if even(x) **then**
        y := 2*y; x := x / 2
    else
        r := r+y; x := x–1
    fi {q}

3. [10 pts] Give a full proof outline obtained by expansion of the partial proof outline below. Work backward though the program (use wp on the four assignments). Show the results of substitutions somewhere.

    {y ≥ 1} x := 0; r := 1;
    {**inv** p ≡ 1 ≤ r = 2^x ≤ y}
    while 2*r ≤ y **do**
        r := 2*r; x := x+1
    od
    {r = 2^x ≤ y ≤ 2^(x+1)}

4. [5 points] For each of the following, say yes or no and explain briefly. If "no", also say whether this is a problem or not and explain briefly.
    a. Does a full formal proof map to a unique full proof outline?
    b. Does a full proof outline map to a unique minimal proof outline?
    c. Does a partial proof outline map to a unique full proof outline?
    d. Does a full proof outline map to a unique full proof?
    e. Which of the following maps to a longer full proof? I.e., one with more lines? (Assume each S is an arbitrary simple statement (an assignment or **skip**).)

        {p_1} S_1 ; {p_2} S_2 ; {p_3} S_3; {p_4} S_4 {p_5} {p_6}
        {q_1} **if** B **then** {q_2} {q_3} S_1; {q_4} S_2 **else** {q_5} S_3 {q_6} **fi** {$q_4 \vee q_6$}
        {r_1} {**inv** r_1} **while** B **do** {r_2} S_1 {r_3} {r_1} **od**; {r_4} S_2 {r_5}

Class 18: Convergence [25 points total]

5. [5 points] For {**inv** p} {**bd** t} **while** B **do** S **od** {$p \wedge \neg B$}, which of the following properties must be hold to get convergence? Briefly discuss why the wrong properties are wrong.
    a. $(p \wedge B \wedge t = t_0) \rightarrow wp(S, t < t_0)$

b. $sp(p \wedge B \wedge t = t_0, S) \rightarrow t < t_0$
c. $p \wedge t > 0 \rightarrow B$
d. $p \wedge \neg B \rightarrow t = 0$
e. $\{p \wedge B \wedge t > t_0\}\ S\ \{t = t_0\}$
f. $p \wedge t = 0 \rightarrow \neg B$

6. [10 = 5 * 2 points] Consider the loop

   {**inv** p} {**bd** t} **while** k ≤ n **do** … k := k+1 **od**

   Assume $p \rightarrow (n \geq 0 \wedge 0 < C \leq k \leq n+C)$ (where C is a named constant). For each of the following expressions, say whether or not it can be used as the bound expression t above (if not, briefly explain why). Include a list of predicate logic obligations and show the expansion of any substitutions.

   a. n – k
   b. n + k + C
   c. n – k + C
   d. n – k + 2*C
   e. 2^(n+C) / 2^k

7. [10 points] Complete the proof of total correctness of the program below by filling in the missing pieces that ensure convergence. You'll have predicates p_0 – p_7 and the bound expression t. If you want, feel free to define other predicates ("Let q ≡ predicate"). Also Include a list of predicate logic obligations and the results of any substitutions. ***Notation***: Below, |b| is a synonym for size(b); use either notation you like.

```
{p₀ ∧ 0 ≤ c < |b|}
x := 1; {p₁} k := 0; {p₂}
{inv p ≡ x = 2^k ≤ b[c] ∧ 0 ≤ c < |b| ∧ p₃}  // [2022-10-31] typo fix
{bd t}                                        // Hint: p₃ ensures that the bound is ≥ 0
while 2*x ≤ b[c] do
    {p ∧ 2*x ≤ b[c] ∧ p₄}
        {p₅} k := k+1; {p₆} x := 2*x
    {p ∧ p₇}
od
{p ∧ 2*x > b[c]}
{x = 2^k ≤ b[c] < 2^(k+1)}
```

Solution to Homework 8

Classes 16 & 17: Proof Outlines

1. (Full outline from formal proof)

```
{ n > 0 }
k := n–1; { n > 0 ∧ k = n–1 }
x := n; { n > 0 ∧ k = n–1 ∧ x = n }
{ inv p } while k > 1 do        // where p ≡ 1 ≤ k ≤ n ∧ x = n!/k!
    { p ∧ k > 1 }
    { p[x*k/x][k–1/k] } k:= k–1;
    { p[x*k/x] } x:= x*k
    { p }
od
{ p ∧ k ≤ 1 }
{ x = n! }
```

2. (Expand partial outline)

```
{ q ≡ r = X*Y–x*y }
if even(x) then
    { q ∧ even(x) } y := 2*y;
    { q₁ ≡ (q ∧ even(x)) [q₀/q] ∧ y = 2*y₀ }      // q₁ ≡ r = X*Y–x*y₀ ∧ even(x) ∧ y = 2*y₀
    x := x / 2
    { q₂ ≡ q₁[x₀/x] ∧ x = x₀ / 2 }                // q₂ ≡ r = X*Y–x₀*y₀ ∧ even(x₀) ∧ y = 2*y₀
else
    { q ∧ odd(x) } r := r+y;
    { q₃ ≡ (q ∧ odd(x))[r₀/r] ∧ r = r₀+y }        // q₃ ≡ r₀ = X*Y–x*y ∧ odd(x) ∧ r = r₀+y
    x := x–1
    { q₄ ≡ q₃[x₀/x] ∧ x = x₀–1 }                  // q₄ ≡ r₀ = X*Y–x₀*y ∧ odd(x₀) ∧ r = r₀+y ∧ x=x₀-1
fi { q₂ ∨ q₄ } { q }
```

3. (Expand partial outline)

```
{ y ≥ 1 }
{ p[1/r][0/x] }                    // p[1/r][0/x] ≡ 1 ≤ 1 = 2^0 ≤ y
x := 0;
{ p[1/r] }                         // p[1/r] ≡ 1 ≤ 1 = 2^x ≤ y
r := 1;
{ inv p ≡ 1 ≤ r = 2^x ≤ y }
while 2*r ≤ y do
    { p ∧ 2*r ≤ y }
    { p[x+1/x][2*r/r] }            // p[x+1/x][2*r/r] ≡ 1 ≤ 2*r = 2^(x+1) ≤ y
    r := 2*r;
    { p[x+1/x] }                   // p[x+1/x] ≡ 1 ≤ r = 2^(x+1) ≤ y
```


x := x+1

{p}

od

{p ∧ 2*r > y}

{r = 2^x ≤ y ≤ 2^(x+1)}

4. (Proofs vs outlines)

   a. A full formal proof does map to a unique full proof outline. Each line of a proof generates one correctness triple, with no choice as to location.

   b. A full proof outline does map to a unique minimal proof outline. Argument is by induction on outline length.

   c. A partial proof outline map can map to multiple unique full proof outlines. A simple example is a sequence of assignments; each one can be expanded using wp or sp, and that choice generates different outlines.

   d. A full proof outline can map can map to multiple unique full proofs. For example, with $\{p_1\}\ \{p_2\}\ S_1;\ \{p_3\}\ S_2\ \{p_4\}$ there's a choice of whether we use precondition strengthening on S_1 or the sequence $S_1; S_2$, and this choice generates different proofs.

   e. (Lengths of proofs) The first proof requires the most number of lines.

      $\{p_1\}\ S_1\ ;\ \{p_2\}\ S_2\ ;\ \{p_3\}\ S_3;\ \{p_4\}\ S_4\ \{p_5\}\ \{p_6\}$

      Requires 9 lines of proof (4 for the individual S_1 – S_4, 3 for the sequences, and 2 for a postcondition weakening of p_5 to p_6).

      $\{q_1\}$ **if** B **then** $\{q_2\}\ \{q_3\}\ S_1;\ \{q_4\}\ S_2\ \{q_7\}$ **else** $\{q_5\}\ S_3\ \{q_6\}$ **fi** $\{q_7 \vee q_6\}$

      Requires 7 lines of proof (1 each for S_1, S_2, and S_3, 2 for a precondition strengthening of q_3 to q_2, 1 for the sequence S_2; S_2, and 1 for the ***if-fi*** statement)

      $\{r_1\}$ {**inv** r_1} **while** B **do** $\{r_2\}\ S_1\ \{r_3\}\ \{r_1\}$ **od**; $\{r_4\}\ S_2\ \{r_5\}$

      Requires 6 lines of proof (1 for S_1, 2 to weaken r_3 to r_1, 1 for the while statement, 1 for S_2, and 1 for the sequence of while loop and S_2.

Class 18: Convergence [25 points total]

5. (Convergence of {**inv** p} {**bd** t} **while** B **do** S **od** {p ∧ ¬B} loop)

   a. Must be true: $(p \wedge B \wedge t = t_0) \rightarrow wp(S, t < t_0)$

   b. Must be true: $sp(p \wedge B \wedge t = t_0, S) \rightarrow t < t_0$

   c. Can be false: $p \wedge t > 0 \rightarrow B$. (t can be > 0 on loop termination)

   d. Can be false: $p \wedge \neg B \rightarrow t = 0$ (Same as previous line: t can be > 0 on loop termination)

   e. Must be true: $\{p \wedge B \wedge t > t_0\}\ S\ \{t = t_0\}$ (Whatever t is at the end of the iteration; it needed to be larger at the start of the iteration.)

   f. Must be true: $p \wedge t = 0 \rightarrow \neg B$ (If t = 0 at the start of an iteration, decreasing it would make t negative at the end of the iteration.)

6. (Possible bound functions for {**inv** p} {**bd** t} **while** $k \le n$ **do** … $k := k+1$ **od** where $p \to (n \ge 0 \wedge 0 < C \le k \le n+C$, for constant C.
   a. $(n - k)$ Cannot be a bound function because it can be negative. Since $k \le n+C$, we can subtract $C+k$ from both sides and get $k - (C+k) \le n+C - (C+k)$, which simplifies to $-C \le n - k$. (Incrementing k does make $n - k$ smaller.)
   b. $n + k + C$ Cannot be a bound function because increasing k makes $n + k + C$ larger, not smaller. (It's nonnegative: $0 < C \le k \le n + C$ implies $0 < n + C$, which implies $k < n + k + C$.)
   c. $n - k + C$ Can be a bound function. Since $k \le n+C$, we know $0 \le n - k + C$, so it's nonnegative, and incrementing k decreases $n - k + C$.
   d. $n - k + 2*C$ Can be a bound function. From part (c), $n - k + C$ is a bound function, and adding a positive constant yields another bound function.
   e. $2 \wedge (n + C) / 2 \wedge k$
      Can be a bound function. It's nonnegative ($0 \le k \le n + C$ implies $2 \wedge k \le 2 \wedge (n + C)$ implies $2 \wedge (n + C) / 2 \wedge k \ge 1$), and it's decreased by incrementing k.
7. (From partial correctness to total correctness.)

   To get a outline for total correctness, we need $p_3 \equiv t \ge 0$, $p_4 \equiv t = t_0$ and $p_7 \equiv t < t_0$. This has implications for p_5 and p_6, but aside from that, everything else comes from the proof of partial correctness.

```
{p₀ ∧ 0 ≤ c < |b|}                // p₀ ≡ b[c] ≥ 1
x := 1;
{p₁}                              // p₁ ≡ b[c] ≥ 1 ∧ 0 ≤ c < |b| ∧ x = 1
k := 0;
{p₂}                              // p₂ ≡ p₁ ∧ k = 0 ≡ b[c] ≥ 1 ∧ 0 ≤ c < |b| ∧ x = 1 ∧ k = 0
{inv p} {bd t}                    // p ≡ x = 2^k ≤ b[c] ∧ 0 ≤ c < |b| ∧ p₃
while 2*x ≤ b[c] do                      where p₃ ≡ t ≥ 0
    {p ∧ 2*x ≤ b[c] ∧ p₄}         // p₄ ≡ t = t₀
    {p₅}                          // p₅ ≡ p₆[k+1 / k] ≡ 2*x = 2^(k+1) ≤ b[c] ∧ t₂ < t₀
    k := k+1;                                where t₄ ≡ t[2*x / x][k+1 / k]
    {p₆}                          // p₆ ≡ (p ∧ p₇)[2*x/x] ≡ 2*x = 2^k ≤ b[c] ∧ t₁ < t₀
    x := 2*x                                 where t₁ ≡ t[2*x / x]
    {p ∧ p₇}                      // p₇ ≡ t < t₀
od
{p ∧ 2*x > b[c]}
{x = 2^k ≤ b[c] < 2^(k+1)}
```