IIT CS536: Science of Programming

Homework 7: Parallelism

Prof. Stefan Muller
TAs: Chaoqi Ma, Zhenghao Zhao

Out: Monday, Apr. 18
Due: Thursday, Apr. 28, 11:59pm CDT

This assignment contains 9 written task(s) for a total of 50 points.

Logistics

Submission Instructions

Please read and follow these instructions carefully.

- Submit your homework on Blackboard under the correct assignment by the deadline (or the extended deadline if taking late days).
- You may submit multiple times, but we will only look at your last submission. Make sure your last submission contains all necessary files.
- Email the instructor and TAs ASAP if
  - You submit before the deadline but then decide to take (more) late days.
  - You accidentally resubmit after the deadline, but did not intend to take late days.

  Otherwise, you do not need to let us know if you're using late days; we'll count them based on the date of your last submission.
- Submit your written answers in a single PDF or Word document. Typed answers are preferred (You can use any program as long as you can export a .pdf, .doc or .docx; LaTeX is especially good for typesetting logic and math, and well worth the time to learn it), but *legible* handwritten and scanned answers are acceptable as well.
- Your Blackboard submission should contain only the file with your written answers. Do not compress or put any files in folders.

Collaboration and Academic Honesty

Read the policy on the website and be sure you understand it.

1 A simple parallel program

Remember our favorite "sum" program? We loved verifying it so much that now we'll verify two of them... in parallel. s_1 calculates the sum of the numbers from 0 to $n/2$ and s_2 calculates the sum from $n/2$ to n. Then, *after that* (remember, $[s_1 \| s_2]$ is just a statement we can sequence with another statement), we add up the two results to get the final sum.

$$s_1 \triangleq \begin{array}{l} i1 := \overline{0}; \\ r1 := \overline{0}; \\ \mathsf{while}\ (i1 < n \mathbin{/} \overline{2})\ \{ \\ \quad r1 := r1 + i1; \\ \quad i1 := i1 + \overline{1} \\ \} \end{array}$$

$$s_2 \triangleq \begin{array}{l} i2 := n \mathbin{/} \overline{2}; \\ r2 := \overline{0}; \\ \mathsf{while}\ (i2 < n)\ \{ \\ \quad r2 := r2 + i2; \\ \quad i2 := i2 + \overline{1} \\ \} \end{array}$$

$$s \triangleq [s_1 \| s_2]; r := r1 + r2$$

Task 1.1 (Written, 4 points).

Show that s from above is a disjoint parallel program by showing that s_1 and s_2 are disjoint. One way to do this is with a table, like in class.

Task 1.2 (Written, 7 points).

Now that we know s is a DPP, we can prove it using the Sequentialize rule. Fill in the blanks in the following proof outline, which is for the program $s_1; s_2; r := r1 + r2$. Recall that for $b \geq a$, $sum(a, b) = a + \cdots + b - 1$.

$$\begin{array}{ll}
 & \{n \geq 0\} \\
i1 := \overline{0}; & \\
r1 := \overline{0}; & \{\underline{\hspace{6cm}}\} \\
\{\mathbf{inv}\ \underline{\hspace{6cm}}\} & \\
\mathsf{while}\ (i1 < n \mathbin{/} \overline{2})\ \{ & \\
\quad r1 := r1 + i1; & \\
\quad i1 := i1 + \overline{1} & \\
\}; & \{\underline{\hspace{6cm}}\} \\
i2 := n \mathbin{/} \overline{2}; & \\
r2 := \overline{0}; & \{\underline{\hspace{6cm}}\} \\
\{\mathbf{inv}\ \underline{\hspace{6cm}}\} & \\
\mathsf{while}\ (i2 < n)\ \{ & \\
\quad r2 := r2 + i2; & \\
\quad i2 := i2 + \overline{1} & \\
\}; & \{\underline{\hspace{6cm}}\} \\
 & \Rightarrow \{r_1 + r_2 = sum(0, n)\} \\
r := r1 + r2 & \\
 & \{r = sum(0, n)\}
\end{array}$$

Task 1.3 (Written, 11 points).

We can also prove this program using the Par rule. Fill in the blanks in the following proof outline with pre- and post-conditions for the two threads, as well as a postcondition for $[s_1 \| s_2]$.

Then, make sure you can use the Par rule to do this proof by showing that the two threads have disjoint conditions. One way to do this is with a table, like we did in class.

	$\{n \geq 0\}$
$[$	$\{p_1 \equiv \underline{\qquad\qquad\qquad\qquad\qquad\qquad}\}$
$i1 := \overline{0};$	
$r1 := \overline{0};$	
$\{\textbf{inv}\ \underline{\qquad\qquad\qquad\qquad\qquad\qquad}\}$	
$\mathsf{while}\ (i1 < n \,/\, \overline{2})\ \{$	
$\quad r1 := r1 + i1;$	
$\quad i1 := i1 + \overline{1}$	
$\}$	$\{q_1 \equiv \underline{\qquad\qquad\qquad\qquad\qquad\qquad}\}$
$\|$	$\{p_2 \equiv \underline{\qquad\qquad\qquad\qquad\qquad\qquad}\}$
$i2 := n \,/\, \overline{2};$	
$r2 := \overline{0};$	
$\{\textbf{inv}\ \underline{\qquad\qquad\qquad\qquad\qquad\qquad}\}$	
$\mathsf{while}\ (i2 < n)\ \{$	
$\quad r2 := r2 + i2;$	
$\quad i2 := i2 + \overline{1}$	
$\}$	$\{q_2 \equiv \underline{\qquad\qquad\qquad\qquad\qquad\qquad}\}$
$];$	$\{\underline{\qquad\qquad\qquad\qquad\qquad\qquad}\} \Rightarrow \{r1 + r2 = sum(0, n)\}$
$r := r1 + r2$	
	$\{r = sum(0, n)\}$

2 A More Realistic Parallel Program

Most low-level threading libraries make joining threads back together a little annoying, so if you were to actually write the program above, you'd probably write it more like this:

$$
\begin{array}{ll}
 & \{n \geq 0 \wedge i1 = 0 \wedge i2 = n/2 \wedge r1 = 0 \wedge r2 = 0\} \\
[& \\
\textsf{while}\ (i1 < n \mathbin{/} \overline{2})\ \{ & \\
\quad r1 := r1 + i1; & \\
\quad i1 := i1 + \overline{1} & \\
\} & \\
\| & \{0 \leq i1 \leq n/2 \wedge i2 = n/2 \wedge r1 = 0 \wedge r2 = 0\} \\
\textsf{while}\ (i2 < n)\ \{ & \\
\quad r2 := r2 + i2; & \\
\quad i2 := i2 + \overline{1} & \\
\}; & \\
\{\mathbf{inv}\ i1 \leq n/2 \wedge r1 = \mathit{sum}(0, i1) \wedge r2 = \mathit{sum}(n/2, n)\} & \\
\textsf{while}\ i1 < n \mathbin{/} \overline{2}\ \{\textsf{skip}\}; & \{r1 = \mathit{sum}(0, n/2) \wedge r2 = \mathit{sum}(n/2, n)\} \\
 & \Rightarrow \{r1 + r2 = \mathit{sum}(0, n)\} \\
r := r1 + r2 & \\
] & \\
 & \{r = \mathit{sum}(0, n)\}
\end{array}
$$

Here:

$$
s_1 \triangleq \begin{array}{l}
\textsf{while}\ (i1 < n \mathbin{/} \overline{2})\ \{ \\
\quad r1 := r1 + i1; \\
\quad i1 := i1 + \overline{1} \\
\}
\end{array}
$$

and

$$
s_2 \triangleq \begin{array}{l}
\textsf{while}\ (i2 < n)\ \{ \\
\quad r2 := r2 + i2; \\
\quad i2 := i2 + \overline{1} \\
\}; \\
\textsf{while}\ i1 < n \mathbin{/} \overline{2}\ \{\textsf{skip}\}; \\
r := r1 + r2
\end{array}
$$

Task 2.1 (Written, 10 points).

Write full proof outlines for s_1 and s_2. We've helped you out with some of the loop invariants and intermediate conditions above.

Task 2.2 (Written, 6 points).

Let s_1^* and s_2^* be the full proof outlines you wrote in the previous task. Explain why these two proof outlines are interference-free (which means we can use the Par-OG rule to prove this program correct!) You don't need to prove it formally, but give a careful explanation of how you know.

3 Even More Realistic with Await

Task 3.1 (Written, 3 points).

Rewrite the program from the previous section, but use an await instead of the second While loop in thread 2.

Task 3.2 (Written, 2 points).

Why might you prefer to use await instead of the loop? **Hint:** Does total correctness hold for the original code with the loop? (If you choose to use this hint, explain why or why not and also why that leads you to prefer the new code.)

4 A Buggy (and threfore even more realistic) Parallel Program

Consider the code from Section 2 again but suppose we didn't include all of the initial conditions in the precondition.

$$
\begin{array}{ll}
 & \{n \geq 0\} \\
[& \\
i1 := \overline{0}; & \\
r1 := \overline{0}; & \\
\mathsf{while}\ (i1 < n \mathbin{/} \overline{2})\ \{ & \\
\quad r1 := r1 + i1; & \\
\quad i1 := i1 + \overline{1} & \\
\} & \\
\| & \{n \geq 0\} \\
i2 := n \mathbin{/} \overline{2}; & \\
r2 := \overline{0}; & \\
\mathsf{while}\ (i2 < n)\ \{ & \\
\quad r2 := r2 + i2; & \\
\quad i2 := i2 + \overline{1} & \\
\}; & \\
\{\mathbf{inv}\ i1 \leq n/2 \wedge r1 = sum(0, i1) \wedge r2 = sum(n/2, n)\} & \\
\mathsf{while}\ i1 < n \mathbin{/} \overline{2}\ \{\mathsf{skip}\}; & \{r1 = sum(0, n/2) \wedge r2 = sum(n/2, n)\} \\
r := r1 + r2 & \\
] & \\
 & \{r = sum(0, n)\}
\end{array}
$$

Let $\sigma = \{n = 26, i1 = 8285, i2 = 283472, r1 = 523752, r2 = 105892\}$ (we haven't initialized some of the variables yet, so they have whatever values where in memory before).

If s is the program immediately above, $|M(s, \sigma)| > 1$.

Task 4.1 (Written, 7 points).

a) Explain how we could interleave the two threads in s, starting with state σ, to terminate in a final state where $r \neq sum(0, 26)$.
Hint 1: There are approximately $3n = 78$ execution paths that will have this behavior, or slightly fewer steps than thread 1 takes total.
Hint 2: The loop invariant for the second loop of thread 2 can no longer be proven to hold when we enter the loop, and indeed, it may not hold. Why?

b) You're testing the code above. Assume that every time you run the program, the processor chooses an interleaving randomly[a]. Also assume that the code always starts in a state, like the one above, that allows the bug from (a) to occur. The expected number of times you'd have to run the program to find the bug in testing is $\frac{\text{\# of possible execution paths}}{\text{\# of incorrect execution paths}}$. Approximating wildly, there are at least $\approx \frac{50!}{25! \cdot 25!} \approx 1.3 \times 10^{14}$ possible terminating execution paths[b] Assuming you have a script that can run the program 1000 times per second, how long would it take to find one of the buggy execution paths you described in part a), in expectation?

c) Now, you get a job with a popular Web service and put the above code into their code base, where it is used by one billion people, who each run it an average of 100 times a day (i.e., the code is run 100 billion $= 10^{11}$ times per day)[c]. How long will it take for a user (any user, not one particular user) to discover the bug, in expectation? Use the same formula from c), the estimate of the number of possible interleavings from c) and your estimate from a).

d) With the above answers, explain why it is good for your job prospects that you took CS 536 and learned about program verification.

[a] this assumption is incredibly false, but that doesn't necessarily make it more likely you'll find the bug in testing

[b] There are also some that diverge, but let's not count those.

[c] If you think this is an unrealistic assumption, think about how many times, e.g., the routine to print a timestamp on a Facebook news feed item is called per day.

5 One more wrap-up question

Task 5.1 (Written, 0 points).

How long (approximately) did you spend on this homework, in total hours of actual working time? Your honest feedback will help us with future homeworks.