Parallel Programs

CS 536: Science of Programming, Spring 2018

Due Tue Apr 24

(~~*Due Mon Apr 23, 11:59 pm. Late if turned in on Tue Apr 24;*~~ *Solution to be posted Wed Apr 25)*

4/19 v2 (overrides 4/18); 4/21: p.2, delay; 4/25 solved, 4/27 pp.3,4; 4/30 p.4; 5/1 p.3

A. Instructions

- You can work together in groups of ≤ 4. Submit your work on Blackboard. Submit one copy, under the name of one person in the group (doesn't matter who). Include the names and A-IDs of everyone in the group (including the submitter) inside that copy.

B. Why?

- Adding parallelism makes program execution more interesting and complicated.
- The proof outlines for disjoint parallel programs with disjoint conditions can be combined in parallel.
- Auxiliary variables enable us to reason using previous state information, but without adding computations.
- Threads using shared variables must avoid interfering with the conditions used by other threads.

C. Outcomes

After this homework, you should be able to

- Recognize legal parallel programs, disjoint parallel programs, and parallel programs with disjoint conditions.
- Determine whether a set of variables is auxiliary for a program and if so, the result of removing them.
- Check for interference between the proof outlines of a shared memory parallel program's threads.

D. Problems [100 points total]

1. [8 points] Which, if any, of the following programs use parallelism illegally? Briefly explain why*.
   a. `[ x := y * y; y := y+1 ‖ while z > w do [z := z/2 ‖ w := w*2] od ]`
   b. `[ x := v ‖ y := x + z ‖ z := x * x ]`
   c. `if a < b then [a := a+b ‖ b := a*b] else [a := a+1 ‖ b := b+2] fi`
   d. `[ [ x := v ‖ y := v * w] ‖ z := x * y ]]`
2. [10 = 8 + 2 points] Let $S \equiv$ `[x := x+5; y := x/2 ‖ z := x/3]` and $\sigma = \{$`x` $= 12\}$. (Note that in the first thread the assignments are done sequentially.) (a) Draw an evaluation graph for $\langle S, \sigma\rangle$ and (b) Give $M(S, \sigma)$.
3. [15 = 3 * 5 points] Write out a table showing, for each pair of triples, their *Change*, *Var*, and *Free* sets and whether the pair are parallel disjoint and/or have disjoint conditions.
   - `{x ≠ y} x := u ; y := u {x = y}`
   - `{v = z} z := z+1 ; v := v+1 {v = z}`, and
   - `{w ≥ u} w := u+1; w := v {w > u}`

* Brief = a sentence or two is enough. Long answers will lose points.

4. [8 points] Briefly compare auxiliary variables to program variables and logical variables.

5. [16 = 10 + 6 points] For the program

```
while y < n do
    u := u * x - v; x := x + f(y); y := y * 2 [4/19]; k := k+1
od
```

a. What are the auxiliary labelings induced by {u}, {v}, {x}, {y}, {k}? (Just give the sets of variables.)

b. Rewrite the program, showing the labeling induced by {v, x} [4/19] (it should be consistent) and show the program that results from removing the labeled variables.

6. [8 points] [4/19] Rewritten: Let $\{p\} < S > \{p'\}$ be totally correct but fail an interference-freedom check vs. condition q. Answer each of the following questions. (For each choice [… | …], say which alternative you select.) Give a brief justification for your answer.

a. Failing the check means that what correctness triple is invalid?

b. The correctness triple check fails for [some | all] states satisfying $p \wedge q$.

c. If the check fails for some particular state $\sigma \vDash p \wedge q$, then interference occurs along [some | every] terminating execution path of $\langle S, \sigma\rangle \rightarrow^* \ldots$.

d. Assume $\{p\} < S > \{p'\}$ and q appear as part of a larger overall proof outline $\{p_0\}\ S'^*\ \{q_0\}$. Then for [some | every | possibly no] state satisfying p_0, execution of S' will invalidate q; if invalidation occurs, it occurs) for [some | every] terminating execution of S'.

7. [12 points] List the interference freedom checks for the following proof outlines.

- $\{q_1\} < S_1 > \{q_2\}$
- $\{$**inv** $p_1\}$ **while** B **do** $\{p_2\} < S_2$**;** $\{p_3\}\ S_3 >$ **od;** $\{p_4\} < S_4 > \{p_5\}$ [4/21]

8. [8 points] Draw a full evaluation graph for the following program, starting in an arbitrary state $\sigma = \{y = \alpha\}$. Indicate deadlocked configurations. Remember, evaluation of an **await** is atomic.

    y := 2; [y := y+4; y := 2*(y+2) $\|$ A]
        where $A \equiv$ **await** y < 9 **then if** y < 5 **then** y := y*3 **fi end**

9. [15 points] Give the set of deadlock conditions for:

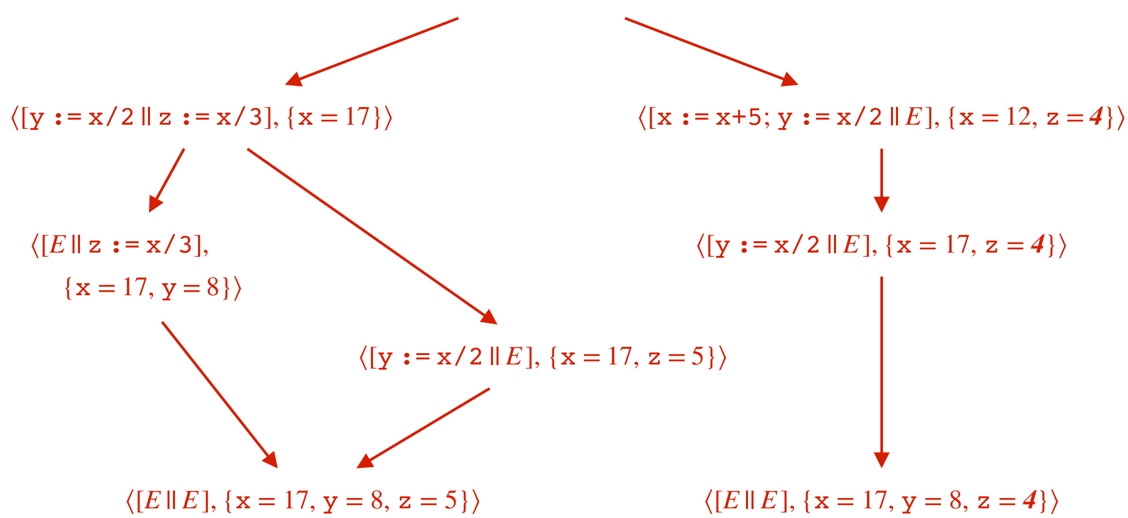
    [$\{p_1\}\ S_1$; $\{q_1\}$ **await** B_1 **then** T_1 **end** $\{r_1\}$
    $\|$ $\{p_2\}$ **await** B_2 **then** S_2 **end;** $\{p_3\}$ **await** B_3 **then** T_3 **end** $\{r_2\}$
    $\|$ $\{p_3\}\ S_3\ \{r_3\}$]

Solution to Homework 6 — Parallel Programs

1. (Legal parallel usage) (a) and (d) are illegal because you can't nest parallel programs.

2. We have $S \equiv$ [x := 18**;** y := x÷2 ‖ z := x÷3] and σ = {x = 12}. From the evaluation graph below, $M(S, \sigma)$ = {{x = 17, y = 8, z = 5}, {x = 17, y = 8, z = 3}}

z = ... values fixed [5/1]



3. The triples are not disjoint parallel but do have disjoint conditions. [4/27] (Free j = 3)

i	*j*	*Change i*	*Vars j*	*Free j*	*Disjoint Pgm?*	*Disjoint Conditions?*
1	2	x y	v z	v z	Yes	Yes
1	3	x y	u v w	u,***w***	Yes	Yes
2	1	v z	u x y	x y	Yes	Yes
2	3	v z	u v w	u,***w***	No	Yes
3	1	w	u x y	x y	Yes	Yes
3	2	w	v z	v z	Yes	Yes

4.
- Program variables appear in the program and can also appear in its proof of correctness.
- Logical variables do not appear in the program, only in the proof of correctness.
- Auxiliary variables are program variables whose values we don't want to calculate and to store in memory; we just want to use them in the proof of correctness.

5. (Auxiliary variables)

5a. Writing ⇒ for "induces", we have {`u`} ⇒ {`u`}; {`v`} ⇒ {`u`, `v`}; {`x`} ⇒ {`u`, `x`}; [4/27]: {`y`} ⇒ {`u`, `x`, `y`} (which is invalid) [4/30]; {`k`} ⇒ {`k`}.

5b. The labeling induced by {`v`, `x`} is {`u`, `v`, `x`}. Adding the labeling to the program gives us

```
while y < n do
    (u) := (u) * x - (v); (x) := (x) + f(y); y := y*2; k := k+1
od
```

To remove the labeled variables, we replace the assignments to (u) and (x) with **`skip`** and eliminate unnecessary **`skip`** statements. The result is

```
while y < n do
    y := y*2; k := k+1
od
```

6. (Failing the interference-freedom check of $\{p\} < S > \{p'\}$ versus q)

a. Failing the check means that $\{p \wedge q\} < S > \{q\}$ is invalid.

b. The correctness triple check fails for *some* state satisfying $p \wedge q$ because invalidity means "not always satisfiable", i.e. $\sigma \not\models \{p \wedge q\} < S > \{q\}$ for some σ.

c. If the check fails for some particular state $\sigma \models p \wedge q$, then interference occurs along *some* terminating execution path of $\langle S, \sigma\rangle \rightarrow^* \ldots$. Since $\sigma \not\models \{p \wedge q\} < S > \{q\}$ iff $\sigma \models p \wedge q$ and $M(S, \sigma) \not\models q$, we know for some $\tau \in M(S, \sigma)$, $\tau \not\models q$. This says nothing about whether any other members of $M(S, \sigma) \models$ or $\not\models q$.

d. Say $\{p\} < S > \{p'\}$ and q occur within $\{p_0\}\ S'^*\ \{q_0\}$. For *possibly no* state satisfying p_0, execution of S' will invalidate q, and if invalidation occurs, it occurs for *some* terminating execution of S'. This is because running S' in a state satisfying p_0 may or may not reach $\{p\} < S > \{p'\}$ at all, much less in a state σ' in which interference with q occurs. Also, if interference is possible when starting in σ', it may or may not occur along every execution path.

7. (Interference freedom checks) For

$\{q_1\} < S_1 > \{q_2\}$

$\{$**`inv`** $p_1\}$ **`while`** B **`do`** $\{p_2\} < S_2$**`;`** $\{p_3\}\ S_3 >$ **`od;`** $\{p_4\} < S_4 > \{p_5\}$

- For $\{q_1\} < S_1 > \{\ldots\}$, we check $\{q_1 \wedge p_j\}\ S_1\ \{p_j\}$ where $j \in \{1, 2, 4, 5\}$, since p_2 and p_4 occur just before atomic statements and p_1 and p_5 are the precondition and postcondition. We should not check p_3 because it occurs *inside* an atomic region.
- In the other direction, we check $\{p_2 \wedge q_j\} < S_2$**`;`** $S_3 > \{q_j\}$ and $\{p_4 \wedge q_j\} < S_4 > \{q_j\}$ where $j \in \{1, 2\}$.

8. (Evaluation graph with possible deadlocks)

$\mathtt{y := 2;\ [y := y+4;\ y := 2*(y+2)} \parallel A]$

where $A \equiv$ **`await`** $\mathtt{y} < 9$ **`then`** **`if`** $\mathtt{y} < 5$ **`then`** $\mathtt{y := y*3}$ **`fi`** **`end`**

$$
\begin{tikzcd}[column sep=small]
 & & \langle \mathtt{y := 2;\ [y := y+4;\ y := 2*(y+2)} \parallel A],\ \{\mathtt{y} = \alpha\}\rangle \arrow[d] & \\
 & & \langle [\mathtt{y := y+4;\ y := 2*(y+2)} \parallel A],\ \{\mathtt{y} = 2\}\rangle \arrow[dl] \arrow[dr] & \\
 & \langle [\mathtt{y := 2*(y+2)} \parallel A],\ \{\mathtt{y} = 6\}\rangle \arrow[dl] \arrow[d] & & \langle [\mathtt{y := y+4;\ y := 2*(y+2)} \parallel E],\ \{\mathtt{y} = 6\}\rangle \arrow[d] \\
\begin{array}{c} \langle [E \parallel A],\ \{\mathtt{y} = 16\}\rangle \\ \text{(deadlocked)} \end{array} & \langle [\mathtt{y := 2*(y+2)} \parallel E],\ \{\mathtt{y} = 6\}\rangle \arrow[d] & & \langle [\mathtt{y := 2*(y+2)} \parallel E],\ \{\mathtt{y} = 10\}\rangle \arrow[d] \\
 & \langle [E \parallel E],\ \{\mathtt{y} = 16\}\rangle & & \langle [E \parallel E],\ \{\mathtt{y} = 24\}\rangle
\end{tikzcd}
$$

9. (Deadlock conditions) In the program below, we have threads with 1, 2, and 0 **`await`** statements, so altogether there are (1+1) * (2+1) * (0+1) – 1 = 5 potential deadlock conditions (assuming no duplicates):

$[\ \{p_1\}\ S_1;\ \{q_1\}$ **`await`** B_1 **`then`** T_1 **`end`** $\{r_1\}$

$\parallel \{p_2\}$ **`await`** B_2 **`then`** S_2 **`end;`** $\{p_3\}$ **`await`** B_3 **`then`** T_3 **`end`** $\{r_2\}$

$\parallel \{p_3\}\ S_3\ \{r_3\}\ ]$

The test conditions are:

- $(q_1 \wedge \overline{B}_1) \wedge (p_2 \wedge \overline{B}_2) \wedge r_3$ — Thread 1 blocked; Thread 2 blocked at 1st **`await`**
- $(q_1 \wedge \overline{B}_1) \wedge (p_3 \wedge \overline{B}_3) \wedge r_3$ — Thread 1 blocked; Thread 2 blocked at 2nd **`await`**
- $(q_1 \wedge \overline{B}_1) \wedge r_2 \wedge r_3$ — Thread 1 blocked
- $r_1 \wedge (p_2 \wedge \overline{B}_2) \wedge r_3$ — Thread 2 blocked at 1st **`await`**
- $r_1 \wedge (p_3 \wedge \overline{B}_3) \wedge r_3$ — Thread 2 blocked at 2nd **`await`**

