Disjoint programs

Last time - parallel pyms \([5_1, 11, 5_2, 11, 5_3]\)

execution interleaved

exec 5_1: one step

exec 5_2: one step

eval graph - multiple final states possible

possible - diverge or interfere along some paths but not of exec along other

problem of understanding parallel

pyms - interaction between threads cause problems

Interference

First step - pyms restricted so that

A interference

pyms exec in parallel, but no
interleaving affects other
threads
Disjoint parallel pmgs

\[ \langle [a := x + 1 \parallel b := x \times 3], \sigma \rangle \]

\[ \langle [E \parallel b := x \times 3], \sigma \rangle \]
\[ \langle [a := x + 1 \parallel E], \sigma [a := \sigma(x) + 1] \rangle \]
\[ \langle [E \parallel E], \sigma [a := \sigma(x) + 1], [b := \sigma(x) \times 3] \rangle \]

The parallelism is limited to pmgs that share read memory but not write memory.

\[ \text{No } \langle [a := x + 1 \parallel c := a + y \parallel a := x \times 3] \]
changes \( a \) \; uses \( a \) \; also changes \( a \)

\( \text{vars changed by a thread} \)
\( \text{vars used by other threads} \)
\( \text{vars used by other threads} = \emptyset \)
\( \text{in } S \) \; \text{used - on \& \& \& 12 hs} \; \text{of } \text{asgt} \)
\( \text{vars}(S) = \text{all vars} \)
\[
\text{changes}(S) = \{ a \text{ vars on lhs of asy } \} \cup S^3
\]

Say \( S \) if \( x > 0 \) then \( y := x^2 \) else \( z := x^3 \).

\[
\text{vars}(S) = \{ x, y, z \} \quad \text{exec uses all 3 vars}
\]

\[
\text{change}(S) = \{ y, z \} \quad \text{even though no exec path changes}
\]

\[
\text{syntactic property} \quad \text{not semantic property}
\]

\( S_1 \) disjoint to \( S_2 \) \iff \text{change}(S_1) \cap \text{vars}(S_2) = \emptyset
\]

\( S_2 \) disjoint to \( S_1 \) doesn't have to be same.

\[ S_1, S_2 \]

\[
\text{change}(S_1) = \{ a^3 \} \quad \text{change}(S_2) = \{ b^3 \}
\]

\[
\text{vars}(S_1) = \{ a, b, c^3 \} \quad \text{vars}(S_2) = \{ b^3 \}
\]

\[
\text{change}(S_1) \cap \text{vars}(S_2) = \emptyset \quad \text{change}(S_2) \cap \text{vars}(S_1) = \emptyset\]

\[
S_1 \cup S_2 \text{ are disjoint} = \{ b^3 \} \neq \emptyset
\]

\( S_1 \) disjoint from \( S_2 \) \iff \( S_1 \cap S_2 = \emptyset \)
<table>
<thead>
<tr>
<th>k</th>
<th>j</th>
<th>Change(S_k)</th>
<th>Vars(S_j)</th>
<th>Sk disj. to S_j?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>a</td>
<td>b</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>b</td>
<td>a,b,c</td>
<td>N</td>
</tr>
</tbody>
</table>

If you have > 2 threads, need each pair of threads to be disjoint.

<table>
<thead>
<tr>
<th>k</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

3 threads
n x (n-1) checks
A parallel is disjoint parallel program (DPP)
if its threads are pairwise disjoint.

DPP models parallel programs where each thread has local write memory and global shared read memory.
No shared write memory.

Without DPP, interleaving is innocuous because no thread asg t affects vars seen by other threads.

Diamond property of DPPs:
If \( \langle s_{t_1}, o_1 \rangle \rightarrow \langle t_{t_1}, o_1 \rangle \)
and \( \langle s_{t_2}, o_2 \rangle \rightarrow \langle t_{t_2}, o_2 \rangle \)
then \( \exists \text{ pgm } t + \text{ state } \exists \in \langle t_{t_1}, o_1 \rangle \rightarrow \langle t_{t_2}, o_2 \rangle \)
and \( \langle t_{t_2}, o_2 \rangle \rightarrow \langle t_{t_2}, o_2 \rangle \)
\(\langle s_{t_1}, o_1 \rangle \rightarrow \langle s_{t_2}, o_1 \rangle \rightarrow \langle s_{t_2}, o_2 \rangle \rightarrow \langle s_{t_2}, o_2 \rangle \)
Then bottom 2 across exist
Diamond property

\[ \langle [S_1 \cup S_2], \sigma \rangle \xrightarrow{\text{solid arrows}} \]

\[ \langle [T_1 \cup T_2], \sigma \rangle \]

Then dashed arrows \[ \langle [T_1 \cup T_2], \tau \rangle \]

Diamond property is very strong - different step of interleaving you get to common pgm/state in 1 step.

Confluence

\[ \langle S_1, \sigma \rangle \]

exec. \[ S_1 \]

any # steps 1 way

\[ \langle T_1, \sigma \rangle \]

any # steps another way

\[ \langle T_2, \sigma \rangle \] 

can get to merge exec. paths (in arithmetic +, -, *, /)

# of steps)

Example of confluent system algebraic formulas eval + substitution
Basically, any calculation system w/o side effects (state dig, errors) is confluent.

\[ 2 \times 3 + 4 \times 5 = 6 \]

Most prog’g langs are not confluent.

Haskell is an exception.

Special case

\[ \langle S, \sigma \rangle \]

\[ \langle E, \sigma_1 \rangle \]

uniquefind

state for

\[ \langle E, \sigma_1 \rangle \]

DPP

\[ \text{exec } S \rightarrow E \text{ one way, get } \sigma_1 \]

\[ \text{exec } S \rightarrow T_2 \rightarrow E, \text{ get } \sigma_0 \text{ again} \]

If \[ T_2 = E \]

then \( \sigma_1 = \sigma_2 \)
Sequentialization proof rule for DPPs

If \([S, 1\ldots S_n] \) is pairwise disjoint and parallel,

then rule

1. \([P_1 P_2 S_1 S_2 \ldots S_n]\) \(\{q_3\}\)
2. \([P_3 [S_1 \ldots S_n] \} q_3\)

Justification - by sequentialization
if pgm is D.P., in particular, every order of exec leads seq exec leads to same result to same result.

W/ a D.P., you can swap any 2 adjacent steps in any evaluation path and not w/ diff. change its result \(S_2 \perp S_1, S_1 \perp S_3\)

threads: \(S_1, S_2 \perp S_3 \perp S_2 \perp S_3 \perp S_2 \perp S_3 \perp S_1\)

By swapping pairs, you can get any ordering (that has all 5 steps in correct order interleaved)

can't swap all 5 steps in incorrect order w/ all 5 steps in correct order
Example $\begin{bmatrix} s_1 \\ s_2 \end{bmatrix}$

\[ a := b \times c; \quad b := a + c; \quad x := 3 \quad \Rightarrow \quad d := c/2; \]

\[ d := d + 1 \]

\[ a := b \times c; \quad b := a + c; \quad x := 3 \quad \Rightarrow \quad d := c/2; \quad d := d + 1 \]

\[ \{ \text{using } \mu_p \} \]

\[ \{ q \} \}

\[ \mu_p \]

\[ a + c \]

\[ c + 1 \]

\[ 3 \]

\[ \{ q \} \]

\[ \{ s_1, s_2 \} \]