A. Why?

- Parallel programs are more flexible than sequential programs but their execution is more complicated.
- Parallel programs are harder to reason about because parts of a parallel program can interfere with other parts.
- Evaluation graphs can be used to show all possible execution paths for a parallel program.

B. Objectives

After this class, you should know

- The syntax and operational & denotational semantics of parallel programs.

C. Basic Definitions for Parallel Programs

- Syntax for parallel statements: $S := [S | | S | | ... | | S]$. We say $[S_1 | | S_2 | | ... | | S_n]$ is the parallel composition of the threads $S_1, S_2, ..., S_n$.
  - The threads must be sequential: You can't nest parallel programs. (You can embed parallel programs within larger programs, such as in the body of a loop.)
  - Example 1: $[x := x+1 | | x := x*2 | | y := x^2]$ is a parallel program with three threads. Since it tries to nest parallel programs, $[x := x+1 | | [x := x*2 | | y := x^2]]$ is illegal.

Interleaving Execution of Parallel Programs

- We run sequential threads in parallel by interleaving their execution. I.e., we interleave the operational semantics steps for the individual threads.
- We execute one thread for some number of operational steps, then execute another thread, etc.
- Depending on the program and the sequence of interleaving, a program can have more than one final state (or cause an error sometimes but not other times).
- As an example, since evaluation of $[x := x+1 | | x := x*2]$ is done by interleaving the operational semantics steps of the two threads, we can either evaluate $x := x+1$ and then $x := x*2$ or evaluate $x := x*2$ and then $x := x+1$.
- The difference between $[x := x+1 | | x := x*2]$ and $if T \rightarrow x := x+1 \ □ T \rightarrow x := x*2 fi$ is that the nondeterministic if-fi executes only one of the two assignments whereas the parallel composition executes both assignments but in an unpredictable order. The sequential nondeterministic if-fi
that simulates the parallel assignments is \[ T \rightarrow x := x + 1; \; x := x^* 2 \; \square \; T \rightarrow x := x^* 2; \; x := x + 1 \; \text{fi}. \] It nondeterministically chooses between the two possible traces of execution for the program.

- Because of the nondeterminism, re-executions of a parallel program can use different orders. For example, two executions of

\[
\text{while } B \text{ do } [x := x + 1 \mid | | x := x^* 2] \text{ od}
\]

can have the same sequence or different sequences of updates to \( x \).

**Difficult to Predict Parallel Program Behavior**

- The main problem with parallel programs is that their properties can be very different from the behaviors of the individual threads.

**Example 2:**

- \( \models \{ x = 5 \} \; x := x + 1 \{ x = 6 \} \) and \( \models \{ x = 5 \} \; x := x^* 2 \{ x = 10 \} \)

- But \( \models \{ x = 5 \} \; x := x + 1 \mid | | x := x^* 2 \{ x = 11 \vee x = 12 \} \)

- The problem with reasoning about parallel programs is that different threads can *interfere* with each other: They can change the state in ways that don’t maintain the assumptions used by other threads.

- Full interference is tricky, so we’re going to work our way up to it. First we’ll look at simple, limited parallel programs that don’t interact at all (much less interfere).

- But before that, we need to look at the semantics of parallel programs more closely.

**D. Semantics of Parallel Programs**

- To execute the sequential composition \( S_1 ; \ldots ; S_n \) for one step, we execute \( S_1 \) for one step.

- To execute the parallel composition \( [S_1 \mid | | \ldots \mid | | S_n] \) for one step, we take one of the threads and evaluate it for one step.

**Operational and Denotational Semantics of Parallel Programs**

- **Definition:** Given \( [S_1 \mid | | \ldots \mid | | S_n] \), for each \( k = 1, 2, \ldots, n \), if \( \langle S_k, \sigma \rangle \rightarrow \langle T_k, \tau_k \rangle \), then

\[
\langle [S_1 \mid | | \ldots \mid | | S_n], \sigma \rangle \rightarrow \langle [S_1 \mid | | \ldots \mid | | S_{k-1} \mid | | T_k \mid | | S_{k+1} \mid | | \ldots \mid | | S_n], \tau_k \rangle
\]

- We’ll write \( E \) for the completely-executed program, so a completely-executed parallel program looks like \([E] \mid | | | [E] \mid | | E\). For consistency's sake we'll treat \([E] \mid | | | [E] \mid | | E\) as \( = E \).

**The \( \rightarrow^* \) Notation**

- **Notation:** The \( \rightarrow^* \) notation has the same meaning whether the configurations involved have parallel programs or not: \( \rightarrow^* \) means \( \rightarrow^n \) for some \( n \geq 0 \), where \( C_0 \rightarrow^n C_n \) means that there is actually a sequence of \( n + 1 \) configurations, \( C_0 \rightarrow C_1 \rightarrow \ldots \rightarrow C_{n-1} \rightarrow C_n \) where we've omitted writing the intermediate configurations.

\* This trick doesn't scale up well to larger programs, but it helps with initially understanding parallel execution.

---

*CS Dept., Illinois Institute of Technology – 2 – © James Sasaki, 2021*
Common Mistake: Writing \( \langle [E \mid |E], \tau \rangle \rightarrow \langle E, \tau \rangle \) is a common mistake. Since there's no transition for the empty program, we can say \( \langle E, \tau \rangle \rightarrow ^0 \langle E, \tau \rangle \) (in 0 steps, nothing happens). With parallel programs, since \([E \mid |E] = E\), we can write \( \langle [E \mid |E], \tau \rangle \rightarrow ^0 \langle E, \tau \rangle \) but not \( \langle [E \mid |E], \tau \rangle \rightarrow \langle E, \tau \rangle \).

Evaluation Graph and Denotational Semantics

- Recall that the evaluation graph for \( \langle S, \sigma \rangle \) is the directed graph of configurations and evaluation arrows leading from \( \langle S, \sigma \rangle \).
- When drawing evaluation graphs, the configuration nodes need to be different.
  - (I.e., if the same configuration appears more than once, show multiple arrows into it — don’t repeat the same node.)
- An evaluation graph shows all possible executions.
  - A program with \( n \) threads will have \( n \) out-arrows from its configuration.
- A path through the graph corresponds to an possible evaluation of the program.
- The denotational semantics of a program in a state is the set of all possible terminating states (plus possibly the pseudostates \( \bot_d \) and \( \bot_e \)). I.e., the states found in the sinks (i.e., at the leaves) of an evaluation graph. (We’ll modify this definition when we get to deadlocked programs.)
  - \( M(S, \sigma) = \{ \tau \in \Sigma \mid \langle S, \sigma \rangle \rightarrow ^* \langle E, \tau \rangle \} \)
    \( \cup \{ \bot_d \} \) if \( S \) can diverge; i.e., if \( \langle S, \sigma \rangle \rightarrow ^* \langle \bot_d, \tau \rangle \) is possible
    \( \cup \{ \bot_e \} \) if \( S \) can produce a runtime error; i.e., \( \langle S, \sigma \rangle \rightarrow ^* \langle \bot_e, \tau \rangle \) is possible

Example 3: The evaluation graph below is for the same program as in Example 2, but starting with an arbitrary state \( \sigma \) where \( \sigma(x) = a \). The graph has two sinks for the two possible final states, so \( M([x := x+1 \mid |x := x+2], \sigma) = \{ \sigma[x := 2a+2], \sigma[x := 2a+1] \} \).

\[
\langle [x := x+1 \mid |x := x+2], \sigma \rangle
\]

let \( a = \sigma(x) \) below

\[
\langle [E \mid |x := x+2], \sigma[x \mapsto a+1] \rangle
\]

\[
\langle [x := x+1 \mid |E], \sigma[x \mapsto 2a] \rangle
\]

\[
\langle [E \mid |E], \sigma[x \mapsto 2a+2] \rangle
\]

\[
\langle [E \mid |E], \sigma[x \mapsto 2a+1] \rangle
\]

Example 3
Example 4: For this example, the evaluation graph is for \((x := v || y := v+2 || z := v+2), \sigma)\), where \(\sigma(v) = a\). \(M([x := v || y := v+2 || z := v+2], \sigma) = \{\sigma[x \mapsto a][y \mapsto a+2][z \mapsto 2a]\}\). Note even though the program is nondeterministic, it produces the same result no matter what execution path it uses. (More generally, if \(S\) is parallel, then \(M(S, \sigma)\) can have more than 1 member, but the converse is not true.)
• **Example 5**: If we keep the program from Example 4 but start evaluation in a different state, then the evaluation graph can be different. Here, $\sigma(v) = 6$, and the evaluation graph is a subgraph of the general graph in Example 4. This time, $M([x := v || y := v+2 || z := v+2], \sigma) = \{\sigma[x \mapsto 6][y \mapsto 8][z \mapsto 12]\}$.

![Evaluation Graph Example 5](image)

- **Example 6**: Let $W = x := 0; \textbf{while } x = 0 \textbf{ do } [x := 0 || x := 1] \textbf{ od}$. Then $M(W, \sigma) = \{\sigma[x \mapsto 1], \perp\}$. The problem here is possible divergence, but it only happens if we always choose thread 1 when we have to make the nondeterministic choice of $[x := 0 || x := 1]$. This is definitely unfair behavior, but it's allowed because of the unpredictability of our nondeterministic choices. In real life, we could add a fairness mechanism to ensure that all threads get to evaluate once in a while.

- If each thread is on a separate processor, then the nondeterministic choice corresponds to which processor is fastest, so the possible divergence of the program is a **race condition**, where the correct behavior of a program depends on the relative speed of the processors involved. Here, divergence occurs if processor 1 is always faster than processor 2 (which also includes the possibility that processor 2 has died).
Example 6
Basics of Parallel Programs

CS 536: Science of Programming, Spring 2021

A. Why

- Parallel programs are more flexible than sequential programs but their execution is more complicated.
- Parallel programs are harder to reason about because parts of a parallel program can interfere with other parts.

B. Objectives

At the end of this work you should be able to
- Draw evaluation graphs for parallel programs.

C. Problems

In general, for the problems below, if it helps you with the writing, feel free to define other symbols. ("Let S = some program," for example.)

1. What is the sequential nondeterministic program that corresponds to the program from Example 4, \( x := v || y := v+2 || z := v*2 \). 

2. Let configuration \( C_2 = \langle S_2, \sigma \rangle \) where \( S_2 = \{x := 1 || x := -1\} \).
   a. What is the sequential nondeterministic program that corresponds to \( S_1 \)?
   b. Draw an evaluation graph for \( C_2 \).

3. Repeat Problem 2 on \( C_3 = \langle S_3, \sigma[v \mapsto 0] \rangle \) where \( S_3 = \{x := v+3; v := v*4 || v := v+2\} \). Note that in the second thread, the two assignments must be done with \( y \) first, then \( z \). Because adding 3 and adding 2 are commutative, two of the (normally-different) nodes will merge.

4. Repeat Problem 2 on \( C_4 = \langle S_5, \sigma[v \mapsto \delta] \rangle \) where \( S_4 = \{v := v*y; v := v+\beta || v := v+\alpha\} \). This problem is similar to Problem 3 but is symbolic, and the commutative plus operator has been moved, so the shape of the graph will be different from Problem 3.
5. Let $C_5 = \langle W, \sigma \rangle$ where $W = \langle \textbf{while } x \leq n \textbf{ do } [x := x+1 || y := y*2] \textbf{ od} \rangle$ and let $\sigma$ of $x$, $y$, and $z$ be 0, 1, and 2 respectively. Note the parallel construct is in the body of the loop.
   a. Draw an evaluation graph for $C_5$. (Feel free to say something like "Let $T = ..." for the loop body, to cut down on the writing.
   b. Draw another evaluation graph for $C_5$, but this time, use the $\rightarrow^3$ notation to get a straight line graph. Concentrate on the configurations of the form $\langle W, ... \rangle$.

6. In $[S_1 || S_2 || ... || S_n]$ can any of the threads $S_1$, $S_2$, ..., $S_n$ contain parallel statements? Can parallel statements be embedded within loops or conditionals?

7. Say we know $\{p_1\} S_1 \{q_1\}$ and $\{p_2\} S_2 \{q_2\}$ under partial or total correctness.
   a. In general, do we know how $\{p_1 \land p_2\} [S_1 || S_2] \{q_1 \land q_2\}$ will execute? Explain briefly.
   b. What if $p_1 = p_2$? I.e., if we know $\{p\} S_1 \{q_1\}$ and $\{p\} S_2 \{q_2\}$, then do we know how $\{p\} [S_1 || S_2] \{q_1 \land q_2\}$ will work?
   c. What if in addition, $q_1 = q_2$? I.e., If we know $\{p\} S_1 \{q\}$ and $\{p\} S_2 \{q\}$, do we know how $\{p\} [S_1 || S_2] \{q\}$ will work? (This problem is harder)
   d. For parts (a) – (c), does it make a difference if we use $\lor$ instead of $\land$?
Solution to Practice 22

Class 22: Basics of Parallel Programs

1. Sequential nondeterministic equivalent of \([x := v || y := v+2 || z := v*2]\):
   
   \[
   \text{if } T \rightarrow x := v; y := v+2; z := v*2
   \]
   
   \[
   \text{☐ } T \rightarrow x := v; z := v*2; y := v+2
   \]
   
   \[
   \text{☐ } T \rightarrow y := v+2; x := v; z := v*2
   \]
   
   \[
   \text{☐ } T \rightarrow y := v+2; z := v*2; x := v
   \]
   
   \[
   \text{fi}
   \]

2. (Program \([x := 1 || x := -1 ]; y := y+x])
   
   a. Equivalent sequential nondeterministic program
      
      \[
      \text{if } T \rightarrow x := 1; x := -1 \sqcup T \rightarrow x := -1; x := 1 \text{ fi}
      \]

   b. Evaluation graph for \([x := 1 || x := -1 ]; y := y+x, \sigma]\)
      
      \[
      \langle [x := 1 || x := -1 ]; y := y+x, \sigma \rangle
      \]
      
      \[
      \langle [E || x := -1 ]; y := y+x, \sigma[x \rightarrow 1] \rangle \quad \langle [x := 1 || E ]; y := y+x, \sigma[x \rightarrow -1] \rangle
      \]
      
      \[
      \langle [E || E ]; y := y+x, \sigma[x \rightarrow -1] \rangle \quad \langle [E || E ]; y := y+x, \sigma[x \rightarrow 1] \rangle
      \]
      
      \[
      \langle E, \sigma[x \rightarrow -1] \{ y \rightarrow \sigma(y) - 1 \} \rangle \quad \langle E, \sigma[x \rightarrow 1] \{ y \rightarrow \sigma(y) + 1 \} \rangle
      \]

3. (Program \([v := v+3; v := v*4 || v := v+2] \))
   
   a. Equivalent sequential nondeterministic program
      
      \[
      \text{if } T \rightarrow v := v+3; \text{ if } T \rightarrow v := v*4; v := v+2 \sqcup T \rightarrow v := v+2; v := v*4 \text{ fi}
      \]
      
      \[
      \text{☐ } T \rightarrow v := v+2; v := v+3; v := v*4 \text{ fi}
      \]
b. Evaluation graph for \(< [v := v+3; v := v*4 || v := v+2], \sigma[v \rightarrow 0] >\). Note that two of the execution paths happen to merge, so there are only two final states instead of three.

\[
\begin{align*}
&\langle [v := v+3; v := v*4 || v := v+2], \sigma[v \rightarrow 0] > \\
&\langle [v := v*4 || v := v+2], \sigma[v \rightarrow 3] > \\
&\langle [v := v*4 || E], \sigma[v \rightarrow 2] > \\
&\langle [v := v+3; v := v*4 || v := v+2], \sigma[v \rightarrow 0] > \\
&\langle [v := v*4 || E], \sigma[v \rightarrow 5] > \\
&\langle [v := v*4 || E], \sigma[v \rightarrow 14] > \\
&\langle [v := v*4 || E], \sigma[v \rightarrow 20] >
\end{align*}
\]

4. (Program \([v := v*y; v := v+\beta || v := v+\alpha] >\)).

a. Equivalent sequential nondeterministic program

\[
\begin{align*}
&if T \rightarrow v := v*y; if T \rightarrow v := v+\beta; v := v+\alpha fi \square T \rightarrow v := v+\alpha; v := v*y; v := v+\beta fi
\end{align*}
\]

b. Evaluation graph for \(< [v := v*y; v := v+\beta || v := v+2], \sigma[v \rightarrow \delta] >\)

\[
\begin{align*}
&\langle [v := v*y; v := v+\beta || v := v+2], \sigma[v \rightarrow \delta] > \\
&\langle [v := v+\beta || v := v+\alpha], \sigma[v \rightarrow \delta y] > \\
&\langle [v := v+\beta || E], \sigma[v \rightarrow \delta y + \alpha] > \\
&\langle [v := v+\beta || E], \sigma[v \rightarrow \delta y + \beta] > \\
&\langle [v := v*y; v := v+\beta || E], \sigma[v \rightarrow \delta + \alpha] > \\
&\langle [v := v*y; v := v+\beta || E], \sigma[v \rightarrow \delta + \gamma] > \\
&\langle [E || v := v+\alpha], \sigma[v \rightarrow \delta y + \beta] > \\
&\langle [E || E], \sigma[v \rightarrow \delta y + \beta + \alpha] >
\end{align*}
\]
5. \( \textbf{while } x \leq n \textbf{ do } [x := x+1 \ || \ || y := y*2] \textbf{ od} \), if \( \sigma(x) = 0, \sigma(y) = 1, \) and \( \sigma(n) = 2. \) Below, let \( T = [x := x+1 \ || \ || y := y*2] \) (just to cut down on the writing).

a. A full evaluation graph. Just to be explicit, I wrote \( \sigma[x \mapsto 0][y \mapsto 1] \) below but just \( \sigma \) is fine.

\[
\begin{align*}
&\langle W, \sigma[x \mapsto 0][y \mapsto 1] \rangle \\
&\quad \downarrow \\
&\langle T; W, \sigma[x \mapsto 0][y \mapsto 1] \rangle \\
&\quad \downarrow \\
&\langle [E || y := y*2]; W, \sigma[x \mapsto 1][y \mapsto 1] \rangle \quad \langle [x := x+1 || E]; W, \sigma[x \mapsto 0][y \mapsto 2] \rangle \\
&\quad \downarrow \\
&\langle W, \sigma[x \mapsto 1][y \mapsto 2] \rangle \\
&\quad \downarrow \\
&\langle T; W, \sigma[x \mapsto 1][y \mapsto 2] \rangle \\
&\quad \downarrow \\
&\langle [E || y := y*2]; W, \sigma[x \mapsto 2][y \mapsto 2] \rangle \quad \langle [x := x+1 || E]; W, \sigma[x \mapsto 1][y \mapsto 4] \rangle \\
&\quad \downarrow \\
&\langle W, \sigma[x \mapsto 2][y \mapsto 4] \rangle \\
&\quad \downarrow \\
&\langle T; W, \sigma[x \mapsto 2][y \mapsto 4] \rangle \\
&\quad \downarrow \\
&\langle [E || y := y*2]; W, \sigma[x \mapsto 3][y \mapsto 4] \rangle \quad \langle [x := x+1 || E]; W, \sigma[x \mapsto 2][y \mapsto 8] \rangle \\
&\quad \downarrow \\
&\langle W, \sigma[x \mapsto 3][y \mapsto 8] \rangle \\
&\quad \downarrow \\
&\langle E, \sigma[x \mapsto 3][y \mapsto 8] \rangle
\end{align*}
\]

b. Evaluation graph abbreviated using \( \rightarrow^3 \) notation:

\[
\begin{align*}
&\langle W, \sigma[x \mapsto 0][y \mapsto 1] \rangle \rightarrow^3 \langle W, \sigma[x \mapsto 1][y \mapsto 2] \rangle \rightarrow^3 \langle W, \sigma[x \mapsto 2][y \mapsto 4] \rangle \\
&\rightarrow^3 \langle W, \sigma[x \mapsto 3][y \mapsto 8] \rangle \rightarrow \langle E, \sigma[x \mapsto 3][y \mapsto 8] \rangle
\end{align*}
\]
7. No, in $[S_1 || S_2 || \ldots || S_n]$ the threads cannot contain parallel statements, but yes, parallel statements can be embedded within loops and conditionals.

8. In general, even if $\{p_1\} S_1 \{q_1\}$ and $\{p_2\} S_2 \{q_2\}$ are both valid sequentially, we can't compose them in parallel, even if $p_1 \equiv p_2$ and $q_1 \equiv q_2$. An example is how $\{x > 0\} x := x - 1 \{x \geq 0\}$ is valid but $\{x > 0\} \{x := x - 1 | x := x - 1\} \{x \geq 0\}$ is not. The first $x := x - 1$ to execute ends with $x \geq 0$, which is too weak for the second $x := x - 1$ to work correctly.