Parallel Programs

- A parallel program is trying to run different threads "at the same time". In our language, the syntax of a parallel statement / program with n threads is $S \equiv [S_1 \parallel S_2 \parallel \cdots \parallel S_n]$. We say $[S_1 \parallel S_2 \parallel \cdots \parallel S_n]$ is the **parallel composition** of **threads** $S_1, S_2, \dots, S_n$.
    - Each thread S_i in the composition should be non-parallel and deterministic: it is not legal to wright $S \equiv \big[S_1 \parallel [S_2 \parallel S_3]\big]$. We don't consider the combination of parallel statements and non-deterministic statements.

- Before we formally define the semantics of parallel programs, let's use a simple example to see the difference between sequential, parallel, and nondeterministic programs.

1. Find a post-condition for each of the following valid triples.
    a) $\{x = 5\}\; x \coloneqq x + 1; x \coloneqq x * 2\; \{q\}$
    It is quite easy to see that $x = 12$ is a valid postcondition: we will finish two assignments in the given order. It is almost the strongest postcondition, we only omitted the initial value of x compared to $x_0 = 5 \wedge x_1 = x_0 + 1 \wedge x = x_1 * 2$.

    b) $\{x = 5\}\; \mathbf{if}\; T \to x \coloneqq x + 1 \;\square\; T \to x \coloneqq x * 2\; \mathbf{fi}\; \{q\}$
    Both arms have true guard, so we will execute two branches at the same time with equal probability. Thus, the postcondition is $x = 6 \vee x = 10$. As an aside, the strongest postcondition is $x_0 = 5 \wedge x = x_0 + 1 \vee x_0 = 5 \wedge x = x_0 * 2$.

    c) $\{x = 5\}\; [x \coloneqq x + 1 \parallel x \coloneqq x * 2]\; \{q\}$
    Both threads will be executed "at the same time"; but some thread must be executed faster than the other in real life, and threads will be executed in any possible order. Thus, we might have $x = 12$ if we execute $x \coloneqq x + 1$ first, or we might have $x = 11$ if we execute $x \coloneqq x * 2$ first. Thus, $x = 11 \vee x = 12$ is a valid postcondition here.

- The above example shows the difference between sequential, parallel, and nondeterministic programs.
    - For a sequential statement, we execute each unit statement in the given order.
    - For a nondeterministic $\mathbf{if} - \mathbf{fi}$ statement, we execute each arm at the same time with the same probability.
    - For a parallel statement, all unit statements in the composition will be executed in any possible order. So **parallel statements can be considered as a simulation of nondeterminism**: $[x \coloneqq x + 1 \parallel x \coloneqq x * 2]$ can simulate $\mathbf{if}\; T \to x \coloneqq x + 1;\; x \coloneqq x * 2 \;\square\; T \to x \coloneqq x * 2; x \coloneqq x + 1\; \mathbf{fi}$.

- **Operational semantic** of parallel statements: given $S \equiv [S_1 \parallel S_2 \parallel \cdots \parallel S_n]$, for each $k = 1,2, \dots, n$, if $\langle S_k, \sigma \rangle \to \langle T_k, \tau_k \rangle$, then $\langle [S_1 \parallel S_2 \parallel \cdots \parallel S_n],\; \sigma \rangle \to \langle [S_1 \parallel \cdots \parallel S_{k-1} \parallel T_k \parallel S_{k+1} \parallel \cdots \parallel S_n], \tau_k \rangle$. If we don't have any runtime error or divergence, the execution of S will end with configuration $\langle E \equiv [E \parallel E \parallel \cdots \parallel E], \tau \rangle$.
    - Note that, from each configuration, we can go to at most n configurations in the next step.
    - The notation $\to^*$ and $\to^k$ that we used in non-parallel program still works here.
    - Note that, $E \equiv [E \parallel E \parallel \cdots \parallel E]$. It is a common mistake to write $\langle [E \parallel E \parallel \cdots \parallel E], \tau \rangle \to \langle E, \tau \rangle$; we can write $\langle [E \parallel E \parallel \cdots \parallel E], \tau \rangle \to^0 \langle E, \tau \rangle$ since $E \equiv [E \parallel E \parallel \cdots \parallel E]$.

- Remember that **denotational semantics** of a statement in a state is the set of all possible terminating states (plus possibly the pseudo states $\bot_d$ and $\bot_e$). It is the same for parallel statements.

2. Show the operational sematic for $\langle S, \sigma \rangle$ till the end where $S \equiv [x \coloneqq x+1 \parallel x \coloneqq x*2]$ and $\sigma = \{x = 5\}$. What is $M(S,\sigma)$?

$$
\begin{tikzcd}[column sep=tiny]
& \langle [x \coloneqq x+1 \parallel x \coloneqq x*2\,], \{x=5\} \rangle \arrow[dl] \arrow[dr] & \\
\langle [E \parallel x \coloneqq x*2], \{x=6\} \rangle \arrow[d] & & \langle [x \coloneqq x+1 \parallel E], \{x=10\} \rangle \arrow[d] \\
\langle [E \parallel E], \{x=12\} \rangle & & \langle [E \parallel E], \{x=11\} \rangle
\end{tikzcd}
$$

- $M(S,\sigma) = \big\{\{x=12\},\{x=11\}\big\}$. Since parallel statements can be considered as a simulation of nondeterminism, it is still true that "If $M(S,\sigma)$ contains more than one states, then S is nondeterministic."

- At each configuration there might be more than one configuration that can be the next step, the operational semantics is usually a directed graph instead of a list, we call this graph an **evaluation graph** (as shown in example 2).
  - While drawing an evaluation graph, we need to make sure that:
    1) Each vertex in the graph is a configuration and each configuration is *unique*.
    2) Each directed edge shows one step (or n steps if $\to^n$, or any number steps if $\to^*$) in the evaluation, and we don't allow multi-edge in the graph. In other words, if we go from one configuration to another twice, we only draw this edge once in the graph.
    3) All the possible executions need to be shown in the graph: if a thread composition has n threads, then there can be at most n outcoming edges from that node.

Let's see an example with three threads.

3. Show the operational sematic for $\langle S, \sigma \rangle$ where $S \equiv [x \coloneqq x+1 \parallel x \coloneqq x*2 \parallel x \coloneqq x*2]$ and $\sigma = \{x = 5\}$. Calculate $M(S,\sigma)$.

$$
\begin{tikzcd}[column sep=tiny]
& & \langle [x \coloneqq x+1 \parallel x \coloneqq x*2 \parallel x \coloneqq x*2], \{x=5\} \rangle \arrow[dll] \arrow[d] \arrow[drr] & & \\
\langle [E \parallel x \coloneqq x*2 \parallel x \coloneqq x*2], \{x=6\} \rangle \arrow[d] \arrow[ddr] & & \langle [x \coloneqq x+1 \parallel E \parallel x \coloneqq x*2], \{x=10\} \rangle \arrow[d] \arrow[ddr] & & \langle [x \coloneqq x+1 \parallel x \coloneqq x*2 \parallel E], \{x=10\} \rangle \arrow[ddl] \arrow[d] \\
\langle [E \parallel E \parallel x \coloneqq x*2], \{x=12\} \rangle \arrow[ddr] & & \langle [E \parallel E \parallel x \coloneqq x*2], \{x=11\} \rangle \arrow[ddrr] & & \langle [E \parallel x \coloneqq x*2 \parallel E], \{x=11\} \rangle \arrow[dd] \\
& \langle [E \parallel x \coloneqq x*2 \parallel E], \{x=12\} \rangle \arrow[d] & & \langle [x \coloneqq x+1 \parallel E \parallel E], \{x=20\} \rangle \arrow[d] & \\
& \langle [E \parallel E \parallel E], \{x=24\} \rangle & & \langle [E \parallel E \parallel E], \{x=21\} \rangle & \langle [E \parallel E \parallel E], \{x=22\} \rangle
\end{tikzcd}
$$

- $M(S,\sigma) = \big\{\{x=21\},\{x=22\},\{x=24\}\big\}$

Then let's see an example with loop.

4. Let $W \equiv \textbf{while } x = 0 \textbf{ do } [x \coloneqq 0 \parallel x \coloneqq 1] \textbf{ od}$. Draw evaluation graph for $\langle x \coloneqq 0; W, \sigma\rangle$, and calculate $M(x \coloneqq 0; W, \sigma)$.

$$
\begin{tikzcd}
 & \langle x \coloneqq 0;\ \textbf{while } x = 0 \textbf{ do } [x \coloneqq 0 \parallel x \coloneqq 1] \textbf{ od}, \sigma\rangle \arrow[d] & \\
 & \langle \textbf{while } x = 0 \textbf{ do } [x \coloneqq 0 \parallel x \coloneqq 1] \textbf{ od}, \sigma[x \mapsto 0]\rangle \arrow[d] & \\
 & \langle [x \coloneqq 0 \parallel x \coloneqq 1]; W, \sigma[x \mapsto 0]\rangle \arrow[dl] \arrow[dr] & \\
\langle [E \parallel x \coloneqq 1]; W, \sigma[x \mapsto 0]\rangle \arrow[d] & & \langle [x \coloneqq 0 \parallel E]; W, \sigma[x \mapsto 1]\rangle \arrow[d] \\
\langle [E \parallel E]; W, \sigma[x \mapsto 1]\rangle \arrow[d, "0"] & & \langle [E \parallel E]; W, \sigma[x \mapsto 0]\rangle \arrow[uuul, bend right, "0"'] \\
\langle W, \sigma[x \mapsto 1]\rangle \arrow[d] & & \\
\langle E, \sigma[x \mapsto 1]\rangle & &
\end{tikzcd}
$$

- From the graph we can see that $M(S, \sigma) = \{\perp_d, \sigma[x \mapsto 1]\}$
- We can see that W could diverge on σ: in each iteration if we execute $x \coloneqq 1$ first, then we will end up with $x = 0$ and we will go to another iteration.
- We call $x = 0$ a **race condition** in parallel program, **which is an unwanted condition that might appear after executing a parallel program when we look for total correctness**. In other words, a race condition in parallel program is a condition that might cause divergence / runtime error or doesn't satisfy the postcondition.

5. Does each of the following triples have race conditions?
   a) $\{T\}\ [\, x \coloneqq 0 \parallel x \coloneqq 1]\ \{x > 0\}$
      Yes. We can get $x = 0$ if we execute the first thread after the second thread, it doesn't satisfy the postcondition, so it is unwanted.

   b) $\{T\}\ [\, x \coloneqq 0 \parallel x \coloneqq 1\,]\ \{x \geq 0\}$
      No.

   c) $\{T\}\ [\, x \coloneqq 0 \parallel x \coloneqq 1\,];\ z = 0 \div x\ \{z = 0\}$
      Yes. $x = 0$ is unwanted.

<u>Disjoint Parallel Program</u>

6. Draw the evaluation graph for configuration $\langle [x \coloneqq v \parallel y \coloneqq v * 2 \parallel z \coloneqq v + 1], \sigma \rangle$ where $\sigma(v) = \alpha$, and v, x, y, z are different variables.

$$\langle [x \coloneqq v \parallel y \coloneqq v * 2 \parallel z \coloneqq v + 1], \sigma \rangle$$

$$\langle [E \parallel y \coloneqq v * 2 \parallel z \coloneqq v + 1],\ \sigma[x \mapsto \alpha] \rangle \qquad \langle [x \coloneqq v \parallel\ y \coloneqq v * 2 \parallel E], \sigma[z \mapsto \alpha + 1] \rangle$$

$$\langle [x \coloneqq v \parallel E \parallel z \coloneqq v + 1], \sigma[y \mapsto 2\alpha] \rangle$$

$$\langle [E \parallel E \parallel z \coloneqq v + 1],\ \sigma[x \mapsto \alpha][y \mapsto 2\alpha] \rangle \qquad \langle [E \parallel y \coloneqq v * 2 \parallel E], \sigma[x \mapsto \alpha][z \mapsto \alpha + 1] \rangle$$

$$\langle [x \coloneqq v \parallel E \parallel E], \sigma[y \mapsto 2\alpha][z \mapsto \alpha + 1] \rangle$$

$$\langle [E \parallel E \parallel E], \sigma[x \mapsto \alpha][y \mapsto 2\alpha][z \mapsto \alpha + 1] \rangle$$

- In this example, although the program is parallel (a simulation of nondeterministic), it generates the same state no matter what execution paths it uses. So, it has the same output as $M(x \coloneqq v;\ y \coloneqq v * 2;\ z \coloneqq v + 1, \sigma)$.

- The program in the above example is a **disjoint parallel program**. Disjoint parallel programs model the situation that multiple threads share readable memory but not writable memory. For every variable x in a disjoint parallel program, there are two situations:
  a) All threads can read x and no threads can write x. In example 6, every thread reads v and no thread writes v.
  b) At most 1 thread can read and write x, and other threads can neither read nor write x. In example 6, the first thread writes x so other threads can neither read nor write x.

- We care about disjoint parallel programs because can use the sequential rule (will be introduced in the next lecture) to come up with a state for denotational semantics without overthinking about the execution order (like in Example 6).

- For statement S, we define that:
  - $vars(S) =$ the set of variables in S. (We either read or write these variables in S)
  - $change(S) =$ the set of variables appears on the left-hand side of assignments in S. (We write these variables in S)

  We say thread S_i **interferes** with S_j if $change(S_i) \cap vars(S_j) \neq \emptyset$.

We say threads S_i and S_j are **disjoint**, if $change(S_i) \cap vars(S_j) = change(S_j) \cap vars(S_i) = \emptyset$.

- If for $0 < i \neq j \leq n$, S_i and S_j are disjoint, then we say threads $S_1, S_2, \dots, S_n$ are **pairwise disjoint**, and we say $[S_1 \parallel S_2 \parallel \cdots \parallel S_n]$ is a **disjoint parallel composition** and also a **disjoint parallel program**.

7. Determine whether the following threads are disjoint.
   a. $S_1 \equiv a := a + x$ and $S_2 \equiv y := y + x$
   Yes. $vars(S_1) = \{a, x\}$ and $change(S_2) = \{y\}$, then $vars(S_1) \cap change(S_2) = \emptyset$; $changes(S_1) = \{a\}$ and $vars(S_2) = \{y, x\}$, then $vars(S_2) \cap change(S_1) = \emptyset$.

   b. $S_1 \equiv a := x$ and $S_2 \equiv x := c$
   No, $vars(S_1) = \{a, x\}$ and $change(S_2) = \{x\}$, then $change(S_2) \cap vars(S_1) \neq \emptyset$; thus S_2 interferes with S_1.

   c. $S_1 \equiv x := a + 1$ and $S_2 \equiv x := b + 1$
   No, both S_1 and S_2 write x, so they interfere with each other.

8. Is the program $S \equiv [S_1 \parallel S_2 \parallel S_3]$ a disjoint parallel program? Here, we have
   $S_1 \equiv a := v;\ v := c + b$
   $S_2 \equiv \textbf{if } b > 0 \textbf{ then } b := c * b \textbf{ else } c := c * 2 \textbf{ fi}$
   $S_3 \equiv \textbf{while } d \geq 0 \textbf{ do } d := d \div 2 - c \textbf{ od}$

   - We can come up with the following table:

i	j	$vars(S_i)$	$changes(S_j)$	S_j interferes with S_i?
1	2	a, v, c, b	b, c	Yes
2	1	b, c	a, v	No
1	3	a, v, c, b	d	No
3	1	d, c	a, v	No
2	3	b, c	d	No
3	2	d, c	b, c	Yes

To sum up, S_2 interferes with both S_1 and S_3, thus the program S is not disjoint parallel.