Parallel Programming

Exercise session 9

Today

- Post-Discussion Ex. 8
- Dekker‘s Lock (a deeper look)
- Kahoot
- Q&A all the topics until now

Post-Discussion Ex. 8

Assume there are two Java threads, sharing the variables v, w, x, y and z. The variables r1 and r2 are private. Assume the code these two threads are executing looks as follows:

Thread 1	Thread 2
x=23;	y=42;
r1 = x;	r2 = y;
v = r1;	w = r2;
z = 2;	

Is the result (the values of the variables after both threads finish the execution) always the same or could it depend on the order in which the threads are scheduled?

The result does not depend on the interleaving. Thread 1 will always see r1=23 and thread 2 always r2=42. The reason that this codes outcome does not depend on the interleaving of threads is that there are no conflicting accesses (at least two accesses by different threads to the same shared variable, of which one is a write).

Thread 1	Thread 2
x = 23;	y = 42;
r1 = x;	r2 = y;
v = r1;	w = r2;
y = 2;	z = 2;

How about this piece of code, is the result always the same or does it depend on the scheduling of the threads?

The result does depend on the interleaving. Both threads access the shared variable y and both write to it.
Thus the value of r2 depends on the order of the last write, either r2=42 or r2=2.

Recap: Relations

n-ary relation

$r \subseteq X_1 \times \dots \times X_n$

2-ary (binary) relation

$r \subseteq X_1 \times X_2$

Example ($<$ relation over $\mathbb{N}$):

$< \subseteq \mathbb{N} \times \mathbb{N}$

Relations can have different properties, for example:

Symmetry: $\forall s_0, s_1 \in S : s_0 R s_1 \rightarrow s_1 R s_0$
Reflexivity: $\forall s_0 \in S : s_0 R s_0$
Transitivity: $\forall s_0, s_1, s_2 \in S : s_0 R s_1 \wedge s_1 R s_2 \rightarrow s_0 R s_2$

Antisymmetric : $a \, R \, b \wedge b \, R \, a \Rightarrow a = b$

Show that the relation ”beats” in the game of rock-paper-scissors is not transitive.

We have:
Scissors beats paper
Paper beats rock

If beats were transitive, we would have:
Scissors beats rock

However, this does not hold. Thus, beats is not transitive

Recap: Transitive Closure

$R^1 = R$

$R^{i+1} = R \circ R^i$ (where $\circ$ denotes composition of relations)

$R^+ = \bigcup_{i=1}^{\infty} R^i$

For the set $S : \{a, b, c, d, e, f\}$ and the relation X over $S : (a, b), (c, d), (a, c), (e, f)$ give the transitive closure X^+.

X^+ : (a, b), (c, d), (a, c), (e, f), (a, d)

Program order is the relation given to actions performed by a thread. We write $S1 \rightarrow S2$ to express that the action $S1$ is performed immediately before $S2$. Note that in a loop, each traversal of the loop body would generate new actions. For the (sequential) code below, which of the statements/actions are in program order? What is the transitive closure of the program order relation on the piece of code below?

```
S1: a=23;
S2: x=3;
S3: if (x==3) {
S4:  b += 1;
    } else {
S5:   b *= 2;
S6:   x = 0;
    }
S7: x=4;
```

$$\rightarrow: (S1, S2), (S2, S3), (S3, S4), (S4, S7)$$
$$\rightarrow^{+}: (S1, S2), (S2, S3), (S3, S4), (S4, S7)$$
$$(S1, S3), (S1, S4), (S1, S7), (S2, S4), (S2, S7), (S3, S7)$$

```
int funca() {
  for (int i=0; i<9999; i++) {
    b=3;
  }
  return b;
}
```

How could a compiler optimize funca() so that it still behaves as intended to an ”observer” who is simply calling the function and using the return value?

A “good” compiler will deduce that this function always returns three and that the for-loop can be removed.

Thread 1	Thread 2
x = 1;	y = 1;
r1 = y;	r2 = x;

Output 1: r1=0, r2=1.
Output 2: r1=1, r2=1.

Answer:

$r1 = 0, r2 = 1 : x = 1 \xrightarrow{hb} r1 = y \xrightarrow{hb} y = 1 \xrightarrow{hb} r2 = x$

$r1 = 1, r2 = 1 : x = 1 \xrightarrow{hb} y = 1 \xrightarrow{hb} r1 = y \xrightarrow{hb} r2 = x$

Both orders respect that $x = 1 \xrightarrow{po} r1 = y$ and $y = 1 \xrightarrow{po} r2 = x$.

For the code below, what are the synchronization actions?

```
x = 0;
volatile ready = false;

Thread 1:          Thread 2:
x = 1;             if (ready) {
ready = true;        print(x)
                   }
```

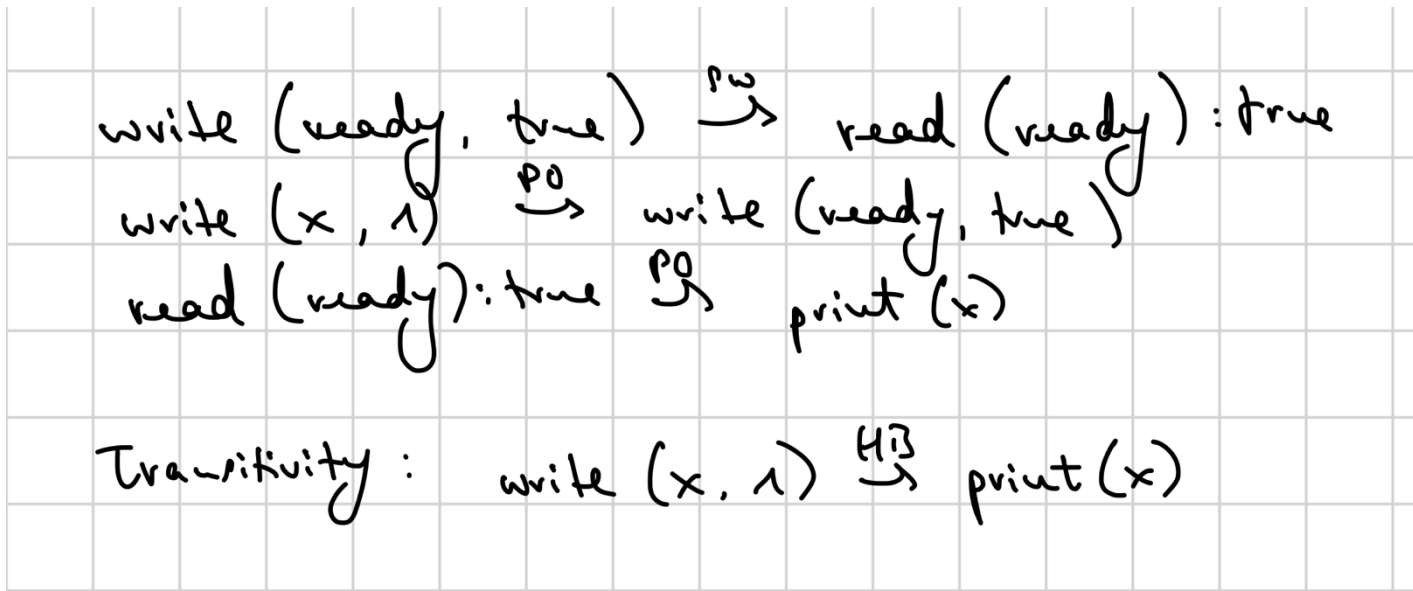
Read and write of volatile variable

When we present such pseudo-code examples we always assume we are seing an excerpt of each threads code, i.e.,
the instructions shown are surrounded by other instructions not relevant for the task.

```
x = 0;
volatile ready = false;

Thread 1:          Thread 2:
x = 1;             if (ready) {
ready = true;        print(x)
                   }
```

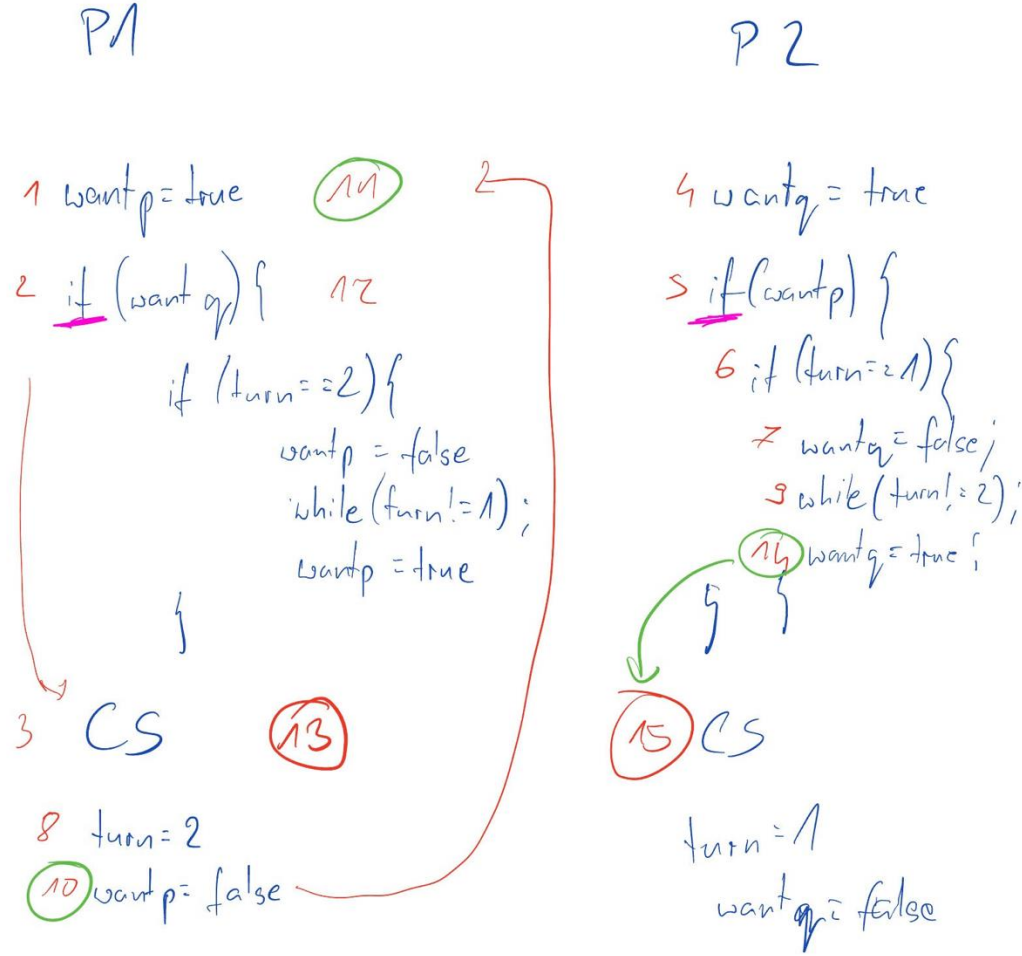
The code in 4.1 has two possible outcomes, according to our memory model: Nothing gets printed or a value gets printed. For the case that something gets printed, explain why it can only be the value 1. Use the happens-before order.



Important

- Program Order, synchronizes-with, etc. are defined on **actions** not program statements

Dekker‘s Lock (a deeper look)



If there was a while then after the step 14
thread P2 would continue looping.
With "if" it moves to its CS.

Kahoot

Request topics