Data Bases II
Exercises

Christian Rossi

Academic Year 2023-2024

Abstract

The course aims to prepare software designers on the effective development of database applications.

First, the course presents the fundamental features of current database architectures, with a specific emphasis on the concept of transaction and its realization in centralized and distributed systems.

Then, the course illustrates the main directions in the evolution of database systems, presenting approaches that go beyond the relational model, like active databases, object systems and XML data management solutions.

Contents

Chapter 1

Exercise session I

Exercise 1

Can the following schedules produce anomalies? c_i and a_i indicate the transactional decision (commit/abort).

1. $r_1(x)w_1(x)r_2(x)w_2(y) \, a_1 \, c_2$

2. $r_1(x)w_1(x)r_2(y)w_2(y) \, a_1 \, c_2$

3. $r_1(x)r_2(x)r_2(y)w_2(y)r_1(z) \, a_1 \, c_2$

4. $r_1(x)r_2(x)w_2(x)w_1(x) \, c_1 \, c_2$

5. $r_1(x)r_2(x)w_2(x)r_1(y) \, c_1 \, c_2$

6. $r_1(x)w_1(x)r_2(x)w_2(x) \, c_1 \, c_2$

Answer of exercise 1

1. We have a serial execution, but with the abort of the first transaction. Since the second transaction reads the modified value of x before the abort, we have a dirty read.

2. We have a serial execution and the two transactions require different resources, so there are no anomalies.

3. There are no anomalies because the last operation of the first transaction works on a different resource.

4. Both transactions first reads in sequence the resource x and then updates it without considering the updated value, so we have a lost update.

2

5. There are no anomalies because the last operation of the first transaction works on a different resource.

6. We have a serial execution, so the schedule is correct.

Exercise 2

The following schedule may produce 2 anomalies: a lost update and a phantom update. Identify them.

$$r_1(x)r_2(x)r_3(x)w_1(x)r_4(y)w_2(x)r_4(x)w_4(y)r_3(y)w_4(x)r_5(y)w_6(y)w_5(y)w_7(y)$$

Answer of exercise 2

We can write the schedule in the following way:

$$
\begin{array}{llllllllllllll}
r_1(x) & & & w_1(x) & & & & & & & & \\
& r_2(x) & & & w_2(x) & & & & & & & \\
& & r_3(x) & & & & & & r_3(y) & & & \\
& & & & & r_4(y) & & r_4(x) & w_4(y) & & w_4(x) & \\
& & & & & & & & & r_5(y) & & w_5(y) \\
& & & & & & & & & & w_6(y) & \\
& & & & & & & & & & & w_7(y)
\end{array}
$$

And we can see that there is a lost update with transactions T_1 and T_2 and a phantom update with T_3 and T_4.

Exercise 3

Classify the following schedule with respect to CRT and VRT classes:
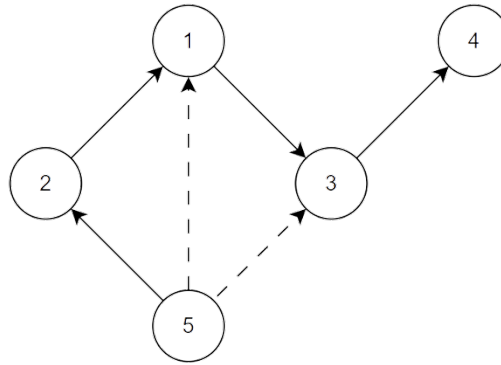
$$r_1(x)r_2(y)w_3(y)r_5(x)w_5(u)w_3(s)w_2(u)w_3(x)w_1(u)r_4(y)w_5(z)r_5(z)$$

Answer of exercise 3

Since CSR contains VSR we check with the conflict graph. To do so we first divide the schedule based on the resources:

- $x : r_1 \, r_5 \, w_3$

- $y : r_2 \, w_3 \, r_4$

- $z : w_5 \, r_5$

- $s : w_3$

- $u : w_5 \, w_2 \, w_1$

The nodes are $\{1, 2, 3, 4, 5\}$ and the arcs are found with the write-write or write-read relations found in the previous groups. So we have the following graph:

Some arcs can be omitted if the nodes are connected in another way (in this case we can remove arcs $\{\{5, 1\}, \{5, 3\}\}$).

There are no cycles: the schedule is CSR (and also VSR).

Exercise 4

Classify the following schedule with respect to CRT and VRT classes:

$$r_2(u)w_2(s)r_1(x)r_2(y)w_3(y)r_5(x)w_5(u)w_3(s)w_2(u)w_3(x)w_1(u)r_4(y)w_5(z)r_5(z)$$

Answer of exercise 4

Since CSR contains VSR we check with the conflict graph. To do so we first divide the schedule based on the resources:

- $x : r_1 \, r_5 \, w_3$

- $y : r_2 \, w_3 \, r_4$

- $z : w_5 \, r_5$

- $s : w_2 \, w_3$

- $u : r_2 \, w_5 \, w_2 \, w_1$

The nodes are $\{1, 2, 3, 4, 5\}$ and the arcs are found with the write-write or write-read relations found in the previous groups. So we have the following graph:
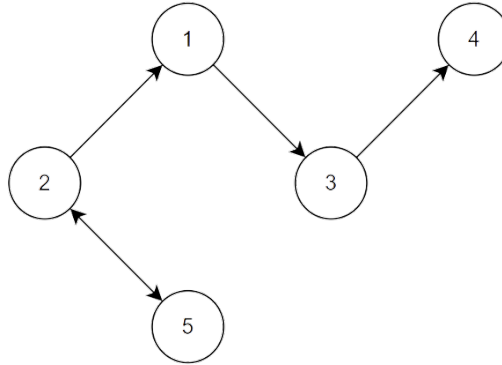
It is possible to see that there is a cycle between two and five. The definition of VSR states that we need to have the same reads-from relations and final writes. So, we try to find a view-equivalent schedule that is also CSR. One possible solution is simply to swap the two writes on the resource u and that is sufficient to eliminate the cycle. So, the schedule:

$$r_2(u)w_2(s)r_1(x)r_2(y)w_3(y)r_5(x)w_5(u)w_2(u)w_3(s)w_3(x)w_1(u)r_4(y)w_5(z)r_5(z)$$

is CSR and also VSR.

Exercise 5

Classify the following schedule with respect to CRT and VRT classes:
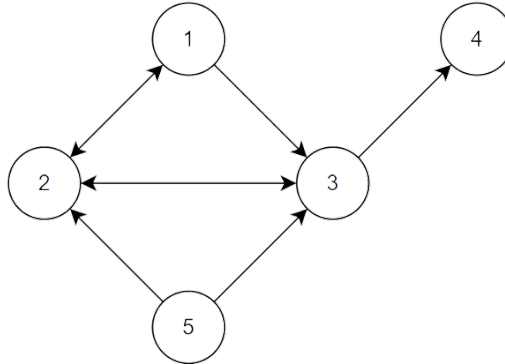
$$r_1(x)r_2(y)w_3(y)r_5(x)w_5(u)w_3(s)w_2(u)w_3(x)w_1(u)r_4(y)w_5(z)r_5(z)r_2(u)w_2(s)$$

Answer of exercise 5

Since CSR contains VSR we check with the conflict graph. To do so we first divide the schedule based on the resources:

- $x : r_1\ r_5\ w_3$

- $y : r_2\ w_3\ r_4$

- $z : w_5\ r_5$

- $s : w_3\ w_2$

- $u : w_5\ w_2\ w_1\ r_2$

The nodes are $\{1, 2, 3, 4, 5\}$ and the arcs are found with the write-write or write-read relations found in the previous groups. So we have the following graph:

In this case it is not possible to find a VSR schedule because it is impossible to do so without changing the final write on s.

Exercise 6

Classify the following schedule with respect to CRT and VRT classes:

$$r_5(x)r_3(y)w_3(y)r_6(t)r_5(t)w_5(z)w_4(x)r_3(z)w_1(y)\ldots$$

$$\ldots r_6(y)w_6(t)w_4(z)w_1(t)w_3(x)w_1(x)r_1(z)w_2(t)w_2(z)$$

Answer of exercise 6

Since CSR contains VSR we check with the conflict graph. To do so we first divide the schedule based on the resources:

- $t:\ r_6\ r_5\ w_6\ w_1\ w_2$

- $x:\ r_5\ w_4\ w_3\ w_1$

- $y:\ r_3\ w_3\ w_1\ r_6$

- $z:\ w_5\ r_3\ w_4\ r_1\ w_2$

The nodes are $\{1, 2, 3, 4, 5, 6\}$ and the arcs are found with the write-write or write-read relations found in the previous groups. So we have the following graph:
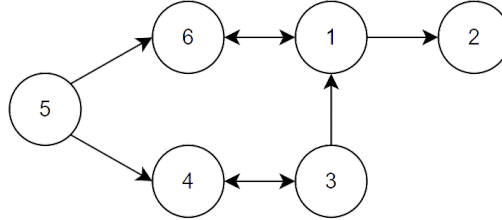
We have two cycles. It is impossible to find a VSR schedule because only the conflict between four and three can be eliminated (the other one changes a read-write relation).

Chapter 2

Exercise session II

Exercise 7

Classify the following schedule with respect to $2PL$ and strict $2PL$ classes:

$$r_1(x)r_2(y)w_3(y)r_5(x)w_5(u)w_3(s)w_2(u)w_3(x)w_1(u)r_4(y)w_5(z)r_5(z)$$
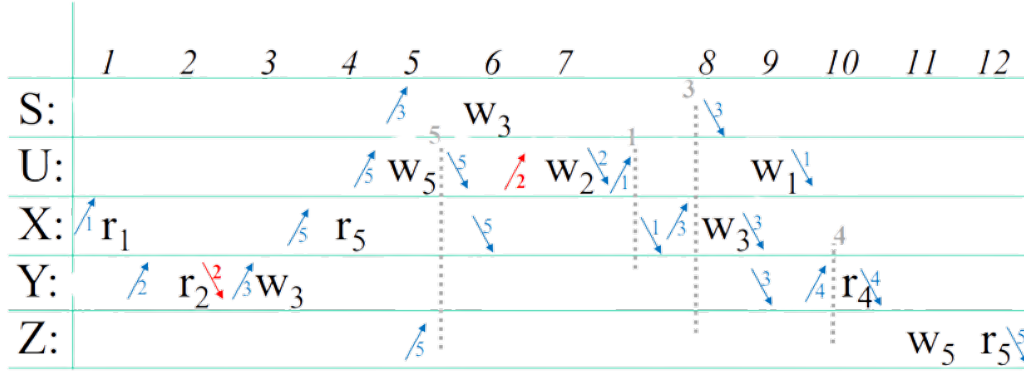
Answer of exercise 7

For strict $2PL$ we assume that all transactions commit and release all locks immediately after their last operation, and check if releases can be executed at commit time.

S clearly cannot be in strict $2PL$. The contradictions are:

- T_1 must release X before 8.

- T_2 must release Y before 7.

- T_5 must release U before 12.

For $2PL$ we have:

	1	2	3	4	5	6	7	8	9	10	11	12
S:					w_3							
U:				w_5		w_2			w_1			
X:	r_1			r_5				w_3				
Y:		r_2	w_3							r_4		
Z:											w_5	r_5

It is also not in $2PL$: an assignment is not possible for T_2 (which must release Y before locking U).

Exercise 8

Classify the following schedule with respect to $2PL$ and strict $2PL$ classes:

$$r_4(x)r_2(x)w_4(x)w_2(y)w_4(y)r_3(y)w_3(x)w_4(z)r_3(z)r_6(z)r_8(z)w_6(z)w_9(z)r_5(z)r_10(z)$$

Answer of exercise 8

For strict $2PL$ we assume that all transactions commit and release all locks immediately after their last operation, and check if releases can be executed at commit time.

$$
\begin{array}{llll}
X: & r_4 \quad r_2 \searrow w_4 & & w_3 \\
Y: & & w_2 \downarrow w_4 \ r_3 & \\
Z: & & & w_4 \ r_3 \ r_6 \ r_8 \ w_6 \ w_9 \ r_5 \ r_{10}
\end{array}
$$

It is therefore clear that the schedule cannot be in $2PL$-strict, due to T_2 and T_4: T_2 ends after 4, but T_4 wants to write X at 3, and T_2 would thus be required to release X earlier, which is impossible if T_2 has to keep all locks until after 4.

For $2PL$ we have:

$$
\begin{array}{llllllllllllllll}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
X: & r_4 & r_2 & w_4 & & & & w_3 & & & & & & & & \\
Y: & & & & w_2 & w_4 & r_3 & & & & & & & & & \\
Z: & & & & & & & & w_4 & r_3 & r_6 & r_8 & w_6 & w_9 & r_5 & r_{10}
\end{array}
$$
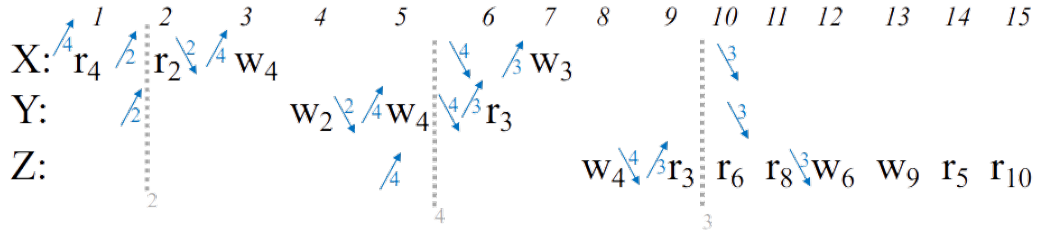
We need to look at those acquisitions that must be anticipated and to those releases that must be delayed to not violate the $2PL$ rules. T_4 can only get the XL on X only after 2 and on Y after 4 and has to release Y before 6 and X before 7. Thus, the lock on Z must be acquired before 6. T_2 can get all the locks at the beginning and release them immediately after each use. T_3 can acquire X, Y and Z just before using them and release them all before 12. All other transactions (T_6, T_9, T_5, T_10) clearly pose no problems.
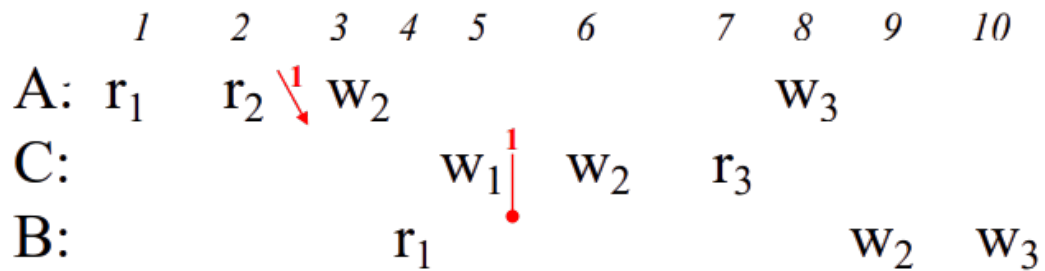
Exercise 9

Classify the following schedule with respect to $2PL$ and strict $2PL$ classes:

$$r_1(A)r_2(A)w_2(A)r_1(B)w_1(C)w_2(C)r_3(C)w_3(A)w_2(B)w_3(B)$$

Answer of exercise 9

For strict $2PL$ we assume that all transactions commit and release all locks immediately after their last operation, and check if releases can be executed at commit time.

The schedule is not strict $2PL$.
For $2PL$ we have:

The schedule is $2PL$.

Exercise 10

Classify the following schedule with respect to $2PL$ and strict $2PL$ classes:

$$r_1(x)w_2(x)r_1(z)w_1(y)r_3(x)r_4(x)w_3(z)w_2(y)r_3(y)w_4(x)w_4(y)$$
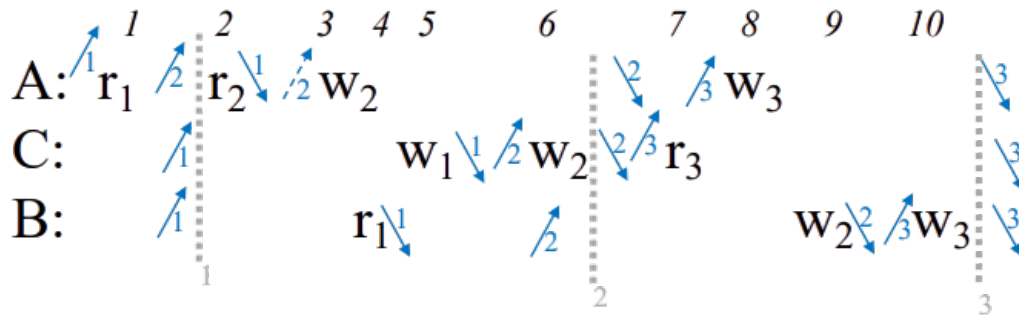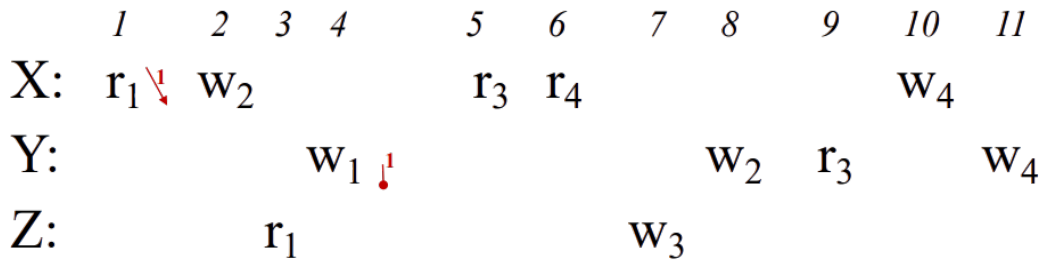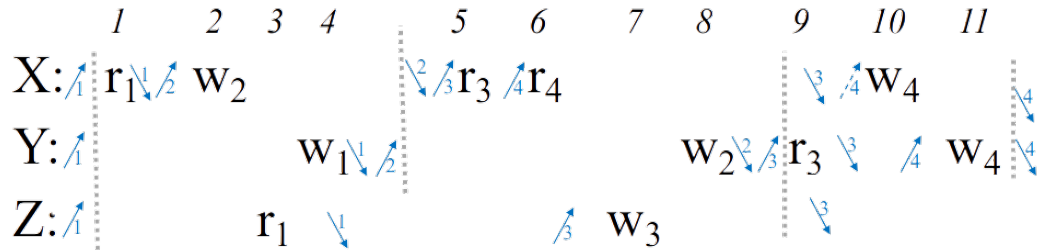
Answer of exercise 10

For strict $2PL$ we assume that all transactions commit and release all locks immediately after their last operation, and check if releases can be executed at commit time.

	1	2	3	4		5	6	7	8	9	10	11
X:	r_1	w_2				r_3	r_4				w_4	
Y:			w_1						w_2	r_3		w_4
Z:			r_1					w_3				

The schedule is not strict $2PL$.
For $2PL$ we have:

	1	2	3	4		5	6	7	8	9	10	11
X:	r_1	w_2				r_3	r_4				w_4	
Y:			w_1						w_2	r_3		w_4
Z:			r_1					w_3				

The schedule is $2PL$.

Exercise 11

Given the schedule:

$$r1(x)r2(x)r3(y)w3(y)w1(x)w2(y)$$

show the sequence of lock and unlock requests produced by the transactions in a $2PL$ execution, in a system with update lock (available locks: SL, UL, XL).

Answer of exercise 11

The locking phases with update locks are the following:

X	Y
$\text{UL}_1(x)$	
$r_1(x)$	
$\text{SL}_2(x)$	
$r_2(x)$	
	$\text{UL}_3(y)$
	$r_3(y)$
	$\text{XL}_3(y)[\text{upgrade}]$
	$w_3(y)$
	$\text{rel}(\text{XL}_3(y))$
	$\text{XL}_2(y)$
$\text{rel}(\text{SL}_2(x))$	
$\text{XL}_1(x)[\text{upgrade}]$	
$w_1(x)$	
$\text{rel}(\text{XL}_1(x))$	
	$w_2(y)$
	$\text{rel}(\text{XL}_2(y))$

Exercise 12

Update lock was introduced to contrast deadlocks. Can we state that deadlocks are impossible in the presence of update locks?

1. If so, concisely explain why.

2. If not, provide a counter-example.

Answer of exercise 12

1. Clearly deadlocks are possible in the presence of UL. Indeed, UL only makes deadlock less likely, by preventing one type of (very frequent) deadlock, due to update patterns, when two transactions compete for the same resource $(r_1(x)r_2(x)w_1(x)w_2(x))$.

2. Consider two distinct resources X and Y, and two transactions that want to access them in this order: $r_1(X)r_2(Y)w_1(Y)w_2(X)$. It is likely that they end up in deadlock, especially if the system on which they run applies $2PL$. UL is totally irrelevant here, because there is no update pattern.

Chapter 3

# Exercise session III

Exercise 13

Consider the following waiting conditions:

- Node A: $E_B \rightarrow t_1, t_1 \rightarrow t_2, E_C \rightarrow t_2, t_2 \rightarrow t_3, t_3 \rightarrow E_B, E_B \rightarrow t_4, t_4 \rightarrow t_3$

- Node B: $E_A \rightarrow t_3, t_3 \rightarrow t_5, t_5 \rightarrow t_6, t_6 \rightarrow E_C, E_C \rightarrow t_7, t_7 \rightarrow t_6, t_9 \rightarrow t_4, t_4 \rightarrow E_A, t_1 \rightarrow E_A$

- Node C: $E_B \rightarrow t_6, t_6 \rightarrow t_8, t_8 \rightarrow t_2, t_2 \rightarrow E_A, t_7 \rightarrow E_B$

Simulate the Obermarck algorithm and indicate whether there is a distributed deadlock.

Answer of exercise 13

We need to construct the graph with the given constraints, that is:

16
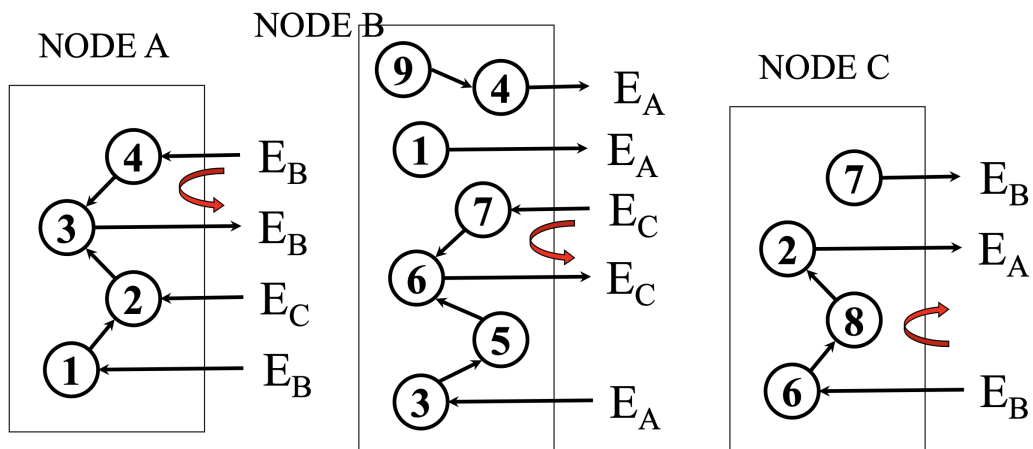
We have to check all the nodes where the sender has a lower value than the receiver. In this case we have that the update is sent to the other distributed node. The interesting cases are highlighted in the image. So, we now have to add the nodes:

- $4 \rightarrow 3$ in E_B.

- $7 \rightarrow 6$ in E_C.

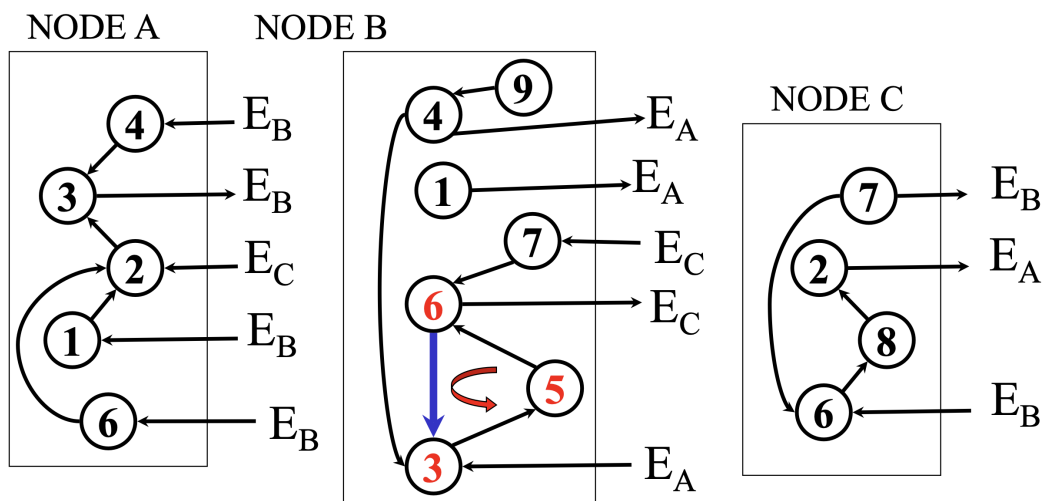- $6 \rightarrow 2$ in E_A.

If the numbered node is not present we can add it to the graph of the distributed node. We obtain the following graphs:
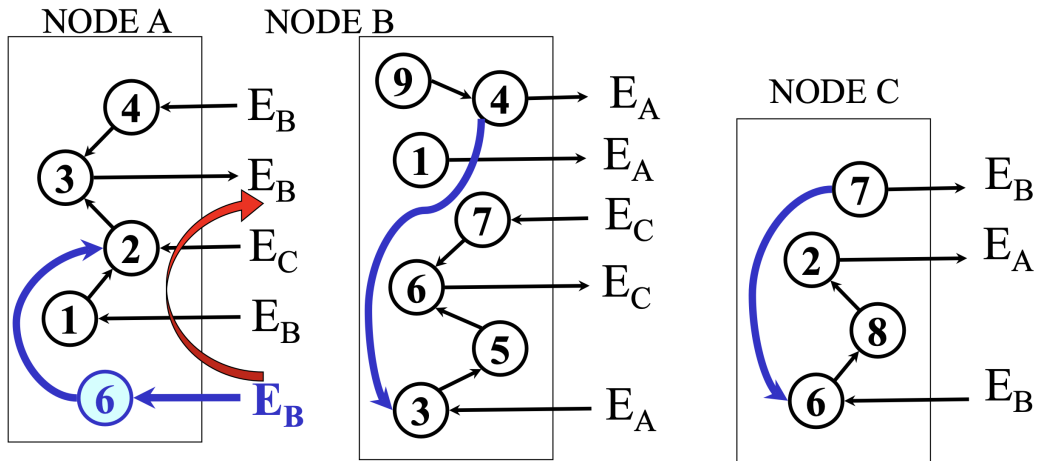
We have to check if other messages are sent. We have:

- $6 \rightarrow 3$ in E_B.

So the updated graph is:

Now we find a cycle, so there is a deadlock.

Exercise 14

The nodes A, B, and C of a distributed transactional system are aware of the following remote and local waiting conditions:

- Node $A : E_B \to t_3, E_C \to t_2, t_1 \to E_C, t_3 \to t_5, t_5 \to t_1$

- Node $B : E_C \to t_2, t_3 \to E_A, t_2 \to t_3$

- Node $C : t_2 \to E_A, t_2 \to E_B, t_1 \to t_4, t_4 \to t_2$

Execute the Obermarck's algorithm twice, with different conventions:

1. Sending messages of the form $E_X \to t_i \to t_j \to E_Y$ forward (toward node Y) and only if and only if $i > j$.

2. With the opposite conventions, so if and only if $i > j$

Discuss the outcome, and explain it, taking into account the properties of the algorithm and the initial conditions.

Answer of exercise 14

The graph is the following:
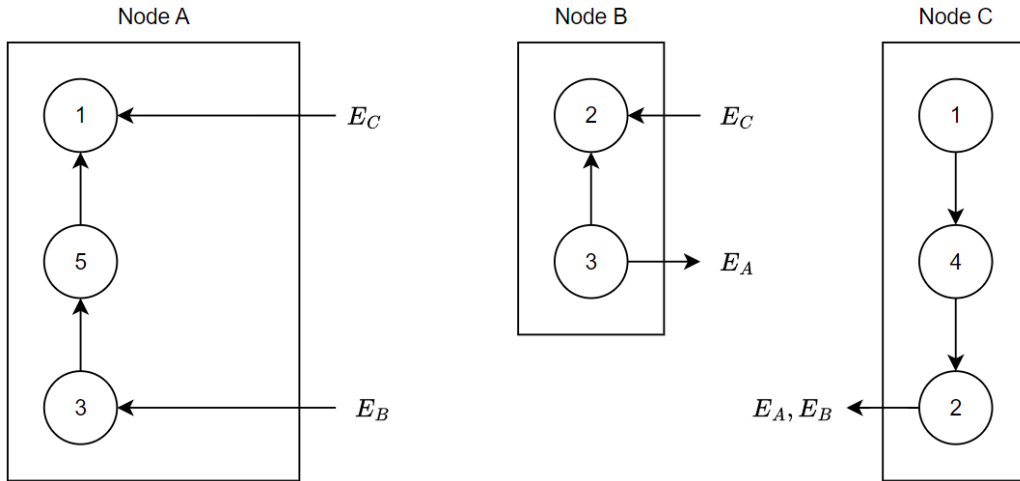
1. We have to add the nodes (connections are distributed):

    - $3 \to 1$ in E_C.
    - $3 \to 2$ in E_A.
    - $3 \to 2$ in E_B.

    By adding those nodes we found that the third one creates a deadlock.

2. We have to add the node (connections are distributed):

   - $2 \rightarrow 3$ in E_A.

   No cycles are found, so no deadlocks found.

The algorithm is independent of the conventions but the initial conventions, but the initial conditions must be consistent and complete. On a faulty dataset even the best algorithm returns untrustworthy results. In this case we have a link missing between node A and C.

Exercise 15

Classify the following schedule with respect to timestamps:

$$r_4(x)r_2(x)w_4(x)w_2(y)w_4(y)r_3(y)w_3(x)w_4(z)r_3(z)r_6(z)r_8(z)w_6(z)w_9(z)r_5(z)r_{10}(z)$$

Answer of exercise 15

We can identify pairs of operations that cause killings:

- $X : r_4 r_2 w_4 w_3$

- $Y : w_2 w_4 r_3$

- $Z : w_4 r_3 r_6 r_8 w_6 w_9 r_5 r_{10}$

On X we have that w_3 is too late with respect to r_4 and w_4. On Y we have that r_3 is late with respect to w_4. On Z we have that r_3 is late with respect to w_4, w_6 with respect to r_8, r_5 with respect to both w_6 and w_9. So, the schedule is not in TS-mono.

The schedule is also outside TS-multi, because $w_3(X)$ comes too late ($r_4(X)$ was already given the initial version instead) and also because $w_6(Z)$ is late with respect to r8(Z). The other five reasons were due to reads (that are always accepted in TS-multi).

Exercise 16

Classify the following schedule with respect to timestamps:

$$r_1(x)r_2(y)w_3(y)r_5(x)w_5(u)w_3(s)w_2(u)w_3(x)w_1(u)r_4(y)w_5(z)r_5(z)$$

Answer of exercise 16

We can identify pairs of operations that cause killings:

- $S : w_3$

- $U : w_5w_2w_1$

- $X : r_1r_5w_3$

- $Y : r_2w_3r_4$

- $Z : w_5r_5$

S, Y and Z are ok, U is ok only if the Thomas rule is applied, and X is not ok for both TS-mono and TS-multi.

Exercise 17

Classify the following schedule:

$$r_1(X)w_1(Y)w_2(Y)w_3(Z)r_1(Z)w_4(X)r_4(Y)w_3(X)r_5(Y)w_5(X)$$

Answer of exercise 17

Firstly, we check if it is CSR:

- $X : r_1w_4w_3w_5$

- $Y : w_1w_2r_4r_5$

- $Z : w_3r_1$

We found a cycle between the nodes three and one, so it is not CSR. It is also not VSR. We now check for TS using the same list: we find that w_4w_3 are not in the correct order, so it is not in TS-mono, neither in TS-multi (it is a write that causes the problem).

Exercise 18

Classify the following schedule:

$$r_1(x)r_2(y)w_3(x)r_5(z)w_6(z)w_2(x)w_3(y)r_7(z)w_4(x)$$

Answer of exercise 19

Firstly, we check if it is CSR:

- $X : r_1 w_3 w_2 w_4$

- $Y : r_2 w_3$

- $Z : r_5 w_6 r_7$

There is a cycle between two and three, so it is not CSR, but by swapping $w_3 w_2$ we can obtain a VSR schedule without changing the schedule.

The schedule is not TS-mono because we have $w_3 w_2$ and so also non TS-multi.

Exercise 19

Given the resource hierarchy below, is the following schedule compatible with a 2PL-strict scheduler that applies hierarchical locking?

$$X$$
$$/ \ \backslash$$
$$A \quad B$$

$$Y$$
$$/ \ \backslash$$
$$S \quad T$$

$$r_1(A)w_1(S)w_2(T)r_2(A)w_1(A)$$

Answer of exercise 18

The lock manager works as follows:

	X	A	B	Y	S	T	
$r_1(A)$	ISL_1	-	-	-	-	-	
	ISL_1	SL_1	-	-	-	-	
$w_1(S)$	ISL_1	SL_1	-	IXL_1	-	-	
	ISL_1	SL_1	-	IXL_1	XL_1	-	
$w_2(T)$	ISL_1	SL_1	-	$IXL_{1,2}$	XL_1	-	
	ISL_1	SL_1	-	$IXL_{1,2}$	XL_1	XL_2	
$r_2(A)$	$ISL_{1,2}$	SL_1	-	$IXL_{1,2}$	XL_1	XL_2	
	$ISL_{1,2}$	$SL_{1,2}$	-	$IXL_{1,2}$	XL_1	XL_2	
commit(T₂)		--- End of T_2
	ISL_1	SL_1	-	IXL_1	XL_1	-	
$w_1(A)$	**IXL_1**	**SL_1**	-	IXL_1	XL_1	-	***Blue**: lock **upgrades***
	IXL_1	**XL_1**	-	IXL_1	XL_1	-	
commit(T₁)		--- End of T_1

So, it is compatible with a strict 2PL scheduler.

25

Exercise 20

Consider the following short schedule occurring on a system with hierarchical lock over a hierarchy where $PagA$ contains tuples t_1 and t_2:

$$r_1(PagA), w_2(t1), w_1(t2)$$

Show a possible sequence of locks, unlocks, lock upgrades, and lock downgrades performed by transactions T_1 and T_2 such that the schedule is in 2PL.

Answer of exercise 20

We have that:

An example of compatible sequence is:	PagA	t2	t1
SIXL1(PagA)	SIXL1	-	-
XL1(t2) *[*]*	SIXL1	XL1	-
r1(PagA)			
U-SL1(PagA) [*lock downgrade*]	IXL1	XL1	-
IXL2(PagA)	IXL1, IXL2	XL1	-
XL2(t1)	IXL1, IXL2	XL1	XL2
w2(t1)			
U-XL2(t1)	IXL1, IXL2	XL1	-
U-IXL2(PagA)	IXL1	XL1	-
Commit(T2)			
w1(t2)		*[*] the "trick" is that T*
U-XL1(t2)	IXL1	*acquires all the locks, a*
U-IXL1(PagA)	-	*t2, even if w_1(t2) is its l*
Commit(T1)		

So the schedule is 2PL.

Exercise 21

Given the resources above and the following transactions:

$$T_1 : r_1(C)w_1(B)w_1(C) \quad T_2 : w_2(A)r_2(C)$$

Consider that T_1 and T_2 can only be scheduled in 10 different ways (two serial, eight interleaved). What can be stated about the 2PL-strict compatibility of these schedules?

Answer of exercise 21

We note that the only potential conflict is between $w_1(C)$ and $r_2(C)$. But these are also the last operations of their respective transactions: to check for compatibility, we can always assume that the commit occurs right after the last operation, and that all locks are released in favor of the other one. But this argument is independent of the order of the operations, and is therefore valid for all the 8 interleaved schedules. So, all the schedules are strict 2PL.