1 Serializabilidad / Recuperabilidad de Historias

1.1 Dada la siguiente historia (schedule) que es conflicto-serializable transformarla en una historia serial mediante una secuencia de intercambios no conflictivos de acciones adyacentes.

H1 = r1(A); w1(A); r2(A); w2(A); r1(B); w1(B); r2(B); w2(B);

T1	T2
r1(A)	
w1(A)	
	r2(A)
	w2(A)
r1(B)	
w1(B)	
	r2(B)
	w2(B)

Orden serial: T1 → T3 es esto lo que pide??

1.2. Dadas las siguientes historias (schedules):

H1 = r1(X); r4(Y); w1(X); w4(X); r3(X); **c1**; w4(Y); w3(Y); w4(Z); **c4**; w3(X); **c3**.
H2 = r1(X); w2(X); r1(Y); w1(X); w1(Y); **c1**; r2(Z); w2(Y); **c2**.
H3 = w1(X); w2(X); w2(Y); **c2**; w1(Y); **c1**; w3(X); w3(Y); **c3.**
H4 = w2(X); r1(X); r2(Z); r1(Y); w2(Y); w1(Y); **c2**; w1(X); **c1**.
H5 = r1(X); r4(Y); r3(X); w1(X); w4(Y); w3(X); **c1**; w3(Y); w4(Z); **c3**; W4(X); **c4**.

(a) Indicar para cada una si es NoRC, RC, ACA o ST.
(b) Indicar cuales podrían producir un dirty read o un lost update.

Historia Recuperable RC

Una historia H es **RC** si siempre que una transacción T_i lee de T_j con $i \neq j$ en H y $c_i \in H$ entonces $c_j < c_i$.
Intuitivamente una historia es recuperable si una transacción realiza commit sólo después de que hicieron commit todas las transacciones de las cuales lee.

Avoids Cascading Aborts ACA

Una historia H es **ACA** si siempre que una transacción T_i lee X de T_j con $i \neq j$ en H entonces $c_j < r_i(X)$.
Lee sólo valores de transacciones que ya hicieron *commit*

Stricta ST

Una historia H es **ST** si siempre que $w_j(X) < o_i(X)$ con $i \neq j$ entonces $a_j < o_i(X)$ o $c_j < o_i(X)$ siendo $o_i(X)$ igual a $r_i(X)$ o a $w_i(X)$
Es decir no se puede leer ni escribir un ítem hasta que la transacción que lo escribió previamente haya hecho *commit* o *abort*.

H1 = r1(X); r4(Y); w1(X); w4(X); r3(X); **c1**; w4(Y); w3(Y); w4(Z); **c4**; w3(X); **c3**.

No es Stricta, ej: w1(X) < w4(X), pero antes de w4 no hay c1 ni a1.
No es ACA, ej: w4(X) < r3(X), pero antes de r3 no hay c4
Parece que es RC, r3(X) se lee de 1 y 4, y tenemos c1 < c4 < c3. Los otros reads no tienen una ti previa.

Podría producirse lost update por la secuencia r1(X), w1(X), w4(X).

H2 = r1(X); w2(X); r1(Y); w1(X); w1(Y); **c1**; r2(Z); w2(Y); **c2**.

No es stricta, ej: w2(x) < w1(x), pero antes de w1(X) no hay c2.
Parece que es ACA, las lecturas no tienen una transacción previa de la cual leer.

Podría producirse lost update por la secuencia r1(X), w2(X), w1(X)

H3 = w1(X); w2(X); w2(Y); **c2**; w1(Y); **c1**; w3(X); w3(Y); **c3.**

No es ST, se tiene w1(X) < w2(X), sin un c1 ni a1 entre ambas.
Parece que es ACA, no hay lecturas,

Podría producirse lost update por la secuencia de writes sin lecturas (??).

H4 = w2(X); r1(X); r2(Z); r1(Y); w2(Y); w1(Y); **c2**; w1(X); **c1**.

No es ST, se tiene w2(X) < r1(X), sin c2 ni a2 en el medio.
No es ACA, se tiene w2(X) < r1(X), sin c2 ni a2 en el medio.
Parece ser RC, c2 < c1 y la lectura r1(Y) no tiene transacción previa

Podría producirse lost update por la secuencia r1(Y), w2(Y), w1(Y)

H5 = r1(X); r4(Y); r3(X); w1(X); w4(Y); w3(X); **c1**; w3(Y); w4(Z); **c3**; W4(X); **c4**.

No es ST: w1(X) < w3(X)
Parece ser ACA.

Podría producirse lost update por la secuencia r1(X), r3(X), w1(X), w3(X)

Y CUALES PUEDEN TENER DIRTY READ? ES SI FALLA EL COMMIT?

1.3. Clasificar según recuperabilidad las siguientes historias:

H1 = r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); **c1**; w3(Y); **c3**; r2(Y); w2(Z); w2(Y); **c2**.

H1 es stricta, (y por lo tanto ACA y RC)

H2 = r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); w3(Y); r2(Y); w2(Z); w2(Y); **c1; c2; c3**.

H2 no es ST y tampoco ACA, pues tenemos w3(Y) < r2(Y) sin un commit 3 en el medio. Ni es RC, porque tenemos que c2 < c3, pero el orden es w3(Y) < r2(Y).

H3 = r1(X); r2(Z); r3(X); r1(Z); r2(Y); r3(Y); w1(X); **c1**; w2(Z); w3(Y); w2(Y); **c3; c2**.

H2 no es ST y tampoco ACA, pues tenemos w3(Y) < r2(Y) sin un commit 3 en el medio. Pero es RC, ya que tenemos el orden de comités c3 < c2.

1.4. Dadas las siguientes transacciones:
T1 = w1(B); r1(C); w(C);
T2 = r2(D) r2(B); w2(C);
(a) Dar una historia H1 que no sea RC.
(b) Dar una historia H2 que sea ACA pero no ST.
(c) Dar una historia H3 que sea ST.

T1 = w1(B); r1(C); w1(C);
T2 = r2(D) r2(B); w2(C);

a) H1 = r2(D); r2(B); w2(C); w1(B); r1(C); w1(C); c1; c2;

No es RC, ya que tenemos w2(C) < r1(C), pero c1 < c2

b) H2 = r2(D); r2(B); w1(B); r1(C); w1(C); w2(C); c2; c1;
Es ACA, pero no ST ya que tenemos w1(C) < w2(C) sin c1 < w2(C)

c) H3 = w1(B); r1(C); w1(C); c1; r2(D); r2(B); w2(C); c2;
Es ST ya que tenemos c1 antes que cualquier operación de la transacción 2.

2 Introducción Protocolos de Bloqueo o Locking

2.1. Dada las siguientes transacciones escritas como Read/Write:
T1 = r1(A); w1(A); r1(B); w1(B)
T2 = r2(A); w2(A); r2(B); w2(B)
Nota: Si bien ambas parecen iguales en realidad pueden realizar operaciones diferentes sobre los ítems leídos.

(a) Incorporar locks para que llevarlas a un modelo con locking binario

T1 = l1(A); r1(A); w1(A); u1(A); l1(B); r1(B); w1(B); u1(B)
T2 = l2(A); r2(A); w2(A); u2(A); l2(B); r2(B); w2(B); u2(B)

(b) Realizar una historia legal pero no serializable

H puede ser:
l1(A); r1(A); w1(A); u1(A); l2(A); r2(A); w2(A); u2(A); l2(B); r2(B); w2(B); u2(B); l1(B); r1(B); w1(B); u1(B)

La transacción 2 se hace antes de que finalice la 1 y como ambas transacciones piden lock sobre A y B, se forma ciclo y no es serializable. Está bien?

2.2. Dadas las siguientes transacciones que deben ser ejecutadas en forma concurrente, indicar una historia legal en el cual se produzca un dirty read:

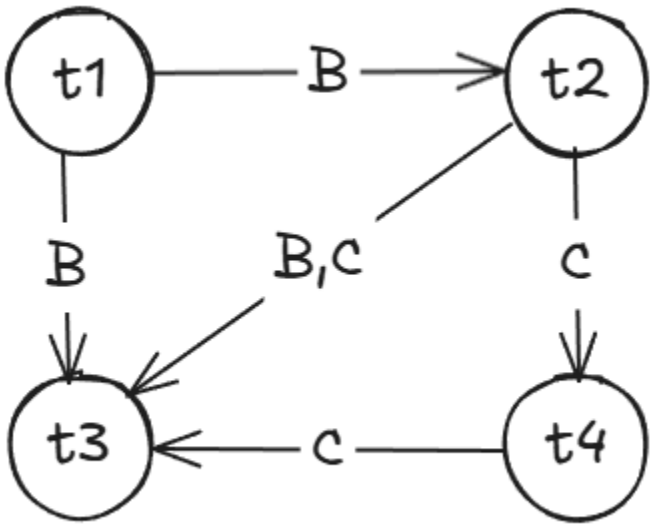
T1 = l1(A); A = A + 2; l1(B); B = A + 5; u1(B); u1(A); l1(C); C = 2 ∗ C; u1(C)
T2 = l2(A); A = A + 1; l2(E); E = A + 8; l2(D); u2(E); D = D/5; u2(A); u2(D)

No es simplemente que mientras se hace alguna escritura falle la otra que escribió antes?

2.3. Considerando la siguiente historia con el modelo introductorio de LOCK/UNLOCK.
(a) Determinar si es serializable.
(b) En caso afirmativo, proponer un schedule serial equivalente

T_1	T_2	T_3	T_4
$l(A)$			
	$l(C)$		
$l(B)$			
	$u(C)$		
$u(A)$			
$u(B)$			
	$l(B)$		
	$u(B)$		
		$l(B)$	
		$u(B)$	
			$l(C)$
			$l(D)$
			$l(E)$
			$u(C)$
		$l(C)$	
			$u(D)$
		$u(C)$	
			$u(E)$

a)



Es serializable, no hay ciclos

b) Un posible orden es T1, T2, T4, T3

2.4. Dadas las transacciones del ejercicio 2.1 llevarlas a un modelo de locking binario tal que ambas sean 2PL.

Two Phase Locking - Definición

Una transacción respeta el protocolo de bloqueo en dos fases (2PL) si todas las operaciones de bloqueo (*lock*) preceden a la primer operación de desbloqueo (*unlock*) en la transacción. Una transacción que cumple con el protocolo se dice que es una ***transacción 2PL***

- Fase de crecimiento: toma los locks
- Fase de contracción: libera los locks

T1 = **l1(A);** r1(A); w1(A); **l1(B);** r1(B); w1(B); **u1(A); u1(B)**
T2 = **l2(A);** r2(A); w2(A); **l2(B);** r2(B); w2(B); **u2(A); u2(B)**

2.5. Dadas las siguientes transacciones:

T1 = l1(A); l1(B); u1(A); u1(B)
T2 = l2(B); l2(A); u2(B); u2(A)

(a) Responder: ¿Cumplen las transacciones T1 y T2 con 2PL?

Sí, todos los locks están antes que cualquier unlock.

(b) Realizar un entrelazado tal que se produzca un deadlock

T1 = **l1(A); l2(B); l2(A); l1(B);** u1(A); u1(B); u2(B); u2(A)

3 Sistemas de Bloqueo con varios modos

3.1. Dadas las siguientes transacciones T1 y T2.
T1 = wl1(A); A = A + 1; wl1(B); B = A + B; u1(A); u1(B)
T2 = rl2(A); wl2(C); C = A + 1; wl2(D); D = 1; u2(A); u2(D); u2(C)
(a) Construir un schedule legal serializable que no sea serial
(b) Responder: ¿Cumplen las transacciones T1 y T2 con 2PL?

a) Serial significa una transacción después de otra? Entonces simplemente me piden un orden tal que sea serializable la historia (que no tenga ciclos el grafo). Como solo hay 2 transacciones es fácil de ver sin dibujar el grafo

T1	T2
wl1(A)	
A=A+1	
wl1(B)	
B=A+B	
u1(A)	
	rl2(A)
u1(B)	
	wl2(C)
	C=A+1
	wl2(D)
	D=1
	u2(A)
	u2(D)
	u2(C)

b) T1 y T2 cumplen 2PL porque tienen todos los locks antes de cualquier unlock.

3.2. Dadas las transacciones T1 , T2 , T3 y T4 con el siguiente entrelazado. Determinar si es serializable y, en caso afirmativo, proponer las correspondientes historias seriales equivalentes.

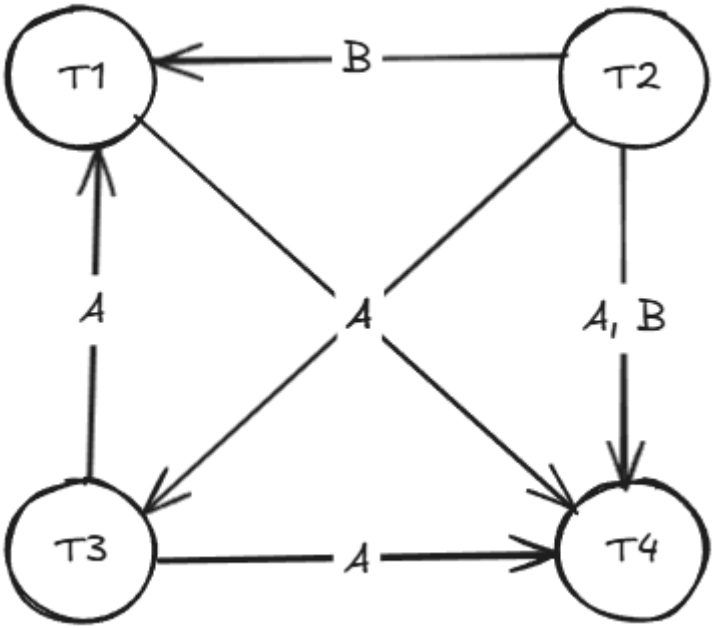
T_1	T_2	T_3	T_4
	$rl(A)$		
		$rl(A)$	
	$wl(B)$		
	$u(A)$		
		$wl(A)$	
	$u(B)$		
$rl(B)$			
		$u(A)$	
			$rl(B)$
$rl(A)$			
			$u(B)$
$wl(C)$			
$u(A)$			
			$wl(A)$
			$u(A)$
$u(B)$			
$u(C)$			

Grafo de precedencia para Locking ternario

1. Hacer un nodo por cada T_i
2. Si T_i hace un $rl_i(X)$ o $wl_i(X)$ y luego T_j con $j \neq i$ hace un $wl_j(X)$ en H hacer un arco $T_i \rightarrow T_j$
3. Si T_i hace un $wl_i(X)$ y T_j con $j \neq i$ hace un $rl_j(X)$ en H entonces hacer un arco $T_i \rightarrow T_j$

Básicamente dice que si dos transacciones realizan un *lock* sobre el mismo ítem y al menos uno de ellas es un *write lock* se debe dibujar un eje desde la primera a la segunda.

No hay ciclos, es serializable.



está bien???

Orden serial: T2, T3, T1, T4.

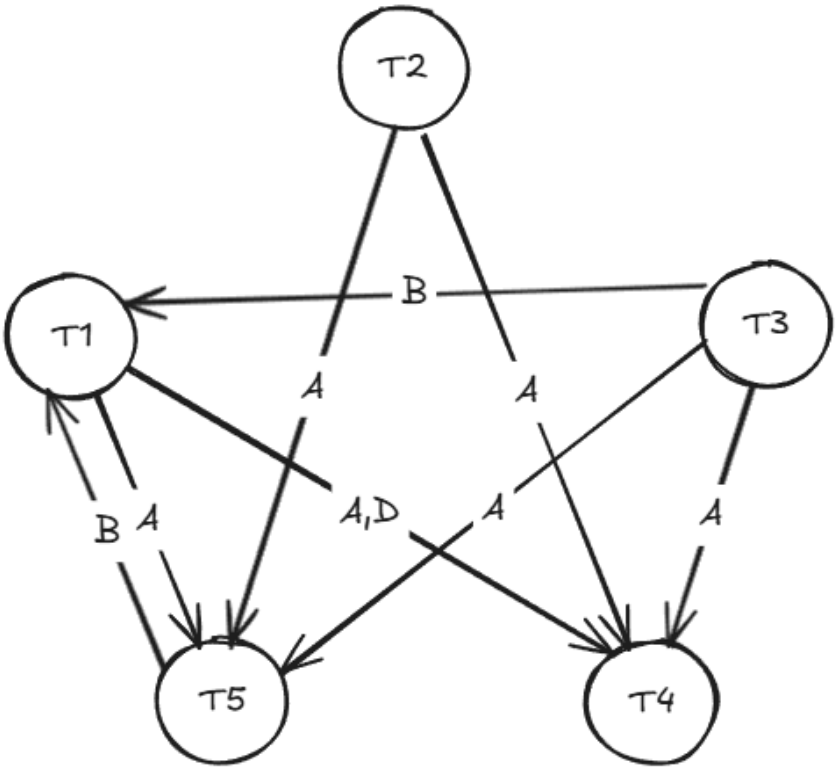
3.3. Dadas las transacciones T1 , T2 , T3 , T4 y T5 con el siguiente entrelazado:
H1 : rl1(A); rl2(A); rl3(A); u1(A); rl5(B); u5(B), wl1(B); wl2(C); u1(B); rl3(B); u3(A); u2(A); wl4(A); u4(A); rl1(D); u1(D); wl4(D); u4(D); wl5(A); u5(A); u2(C); u3(B)
(a) Dibujar el entrelazado en forma tabular.
(b) Determinar si es serializable y, en caso afirmativo, proponer las correspondientes historias seriales equivalentes.
(c) Para cada transacci´on indicar si cumple con ser 2PL. Si no lo son modificarlas para que lo sean.
(d) Sobre el ´ıtem anterior ubicar commits para que sean 2PL estricto y 2PL riguroso.

(a)

T1	T2	T3	T4	T5
rl1(A);				
	rl2(A);			
		rl3(A);		
u1(A);				
				rl5(B);

				u5(B),
wl1(B);				
	wl2(C);			
u1(B);				
		rl3(B);		
		u3(A);		
	u2(A);			
			wl4(A)	
			u4(A);	
rl1(D);				
u1(D);				
			wl4(D);	
			u4(D);	
				wl5(A);
				u5(A);
	u2(C);			
		u3(B)		

b)

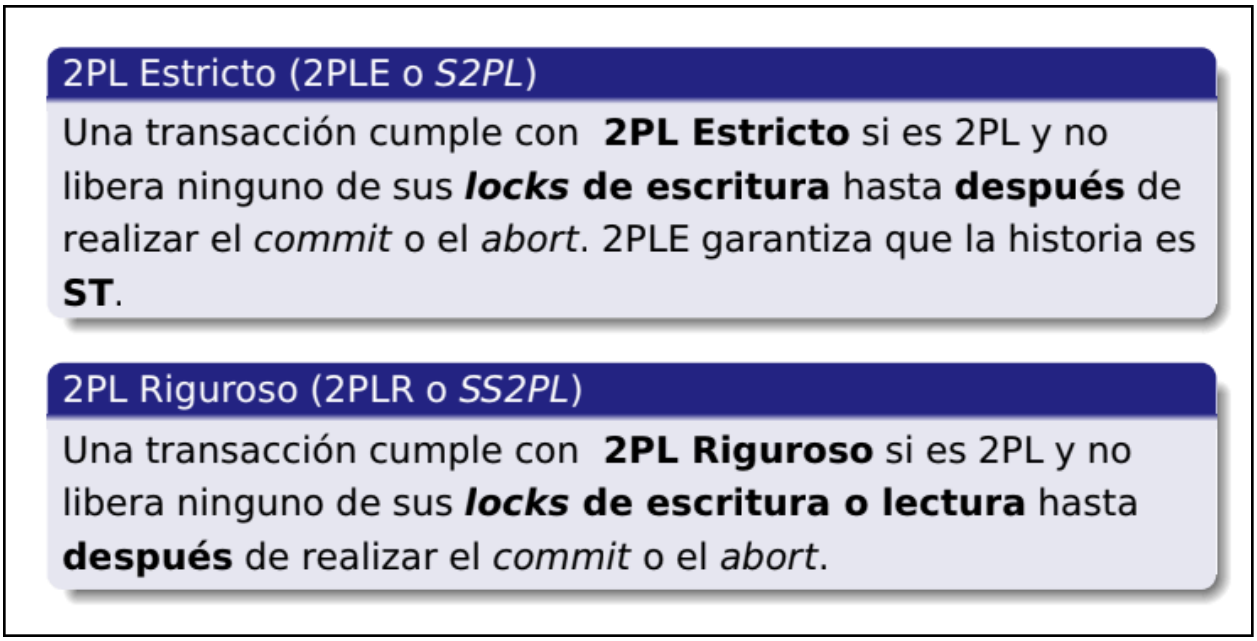


está bien???

No es serializable, hay un ciclo entre T1 y T5.

c) T1, T4 y T5 no es 2PL. T2 y T3 son 2PL.

T1': rl1(A); wl1(B); rl1(D); u1(D); u1(B); u1(A)
T4': wl4(A); wl4(D); u4(A); u4(D)
T5': rl5(B); wl5(A); u5(A) u5(B)



2pl estricto:

T2: rl2(A); wl2(C); u2(A); **C**; u2(C)
T3: rl3(A); rl3(B); u3(A); u3(B); **C**

T1': rl1(A); wl1(B); rl1(D); u1(D); **C**; u1(B); u1(A)
T4': wl4(A); wl4(D); **C**; u4(A); u4(D)
T5': rl5(B); wl5(A); u5(A); **C**; u5(B)

2pl riguroso:

T2: rl2(A); wl2(C); **C**; u2(A); u2(C)
T3: rl3(A); rl3(B); **C**; u3(A); u3(B)

T1': rl1(A); wl1(B); rl1(D); **C**; u1(D); u1(B); u1(A)
T4': wl4(A); wl4(D); **C**; u4(A); u4(D)
T5': rl5(B); wl5(A); **C**; u5(A); u5(B)

3.4. Dadas las transacciones:

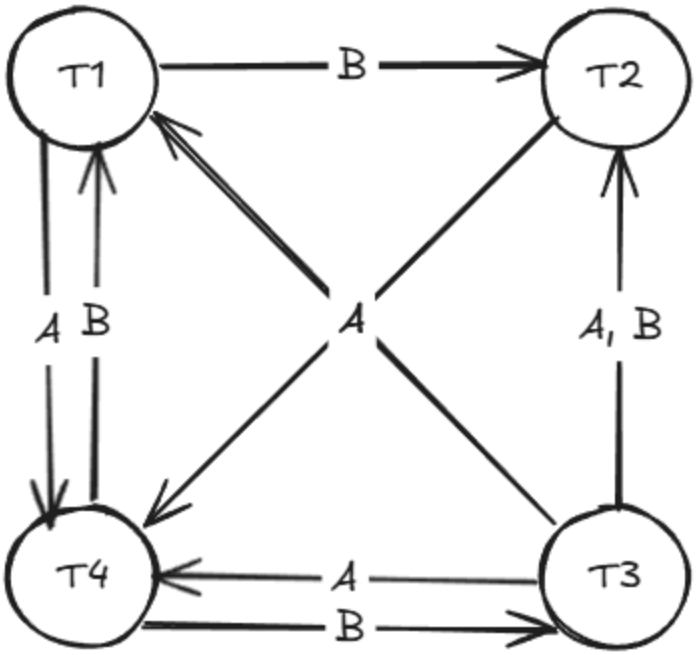
T1 : rl1(A); wl1(B); u1(A); u1(B)
T2 : rl2(A); u2(A); rl2(B); u2(B)
T3 : wl3(A); u3(A); wl3(B); u3(B)
T4 : rl4(B); u4(B); wl4(A); u4(A)

(a) Armar una historia legal y serializable y dar la correspondiente planificación serial.

T1, T2, T3, T4 es esto lo que pide???

b) Dibujar el Grafo de Precedencia de la siguiente historia H1 (SG(H1)) y determinar si es serializable:
H1 : wl3(A); rl4(B); u3(A); rl1(A); u4(B); wl3(B); rl2(A); u3(B); wl1(B); u2(A); u1(A); wl4(A); u1(B); rl2(B); u4(A); u2(B)

T1	T2	T3	T4
		wl3(A)	
			rl4(B)
		u3(A)	
rl1(A)			
			u4(B)
		wl3(B)	
	rl2(A)		
		u3(B)	
wl1(B)			
	u2(A)		
u1(A)			
			wl4(A)
u1(B)			
	rl2(B)		
			u4(A)
	u2(B)		



está bien???

Hay ciclos, no es serializable.

c) Reescribir las transacciones para que adhieran al protocolo 2PL.

T1 : rl1(A); wl1(B); u1(A); u1(B)
T2 : rl2(A); rl2(B); u2(A); u2(B)
T3 : wl3(A); wl3(B); u3(A); u3(B)
T4 : rl4(B); wl4(A); u4(B); u4(A)

3.5. Dadas las transacciones T1 , T2 , T3 con el siguiente entrelazado H1:

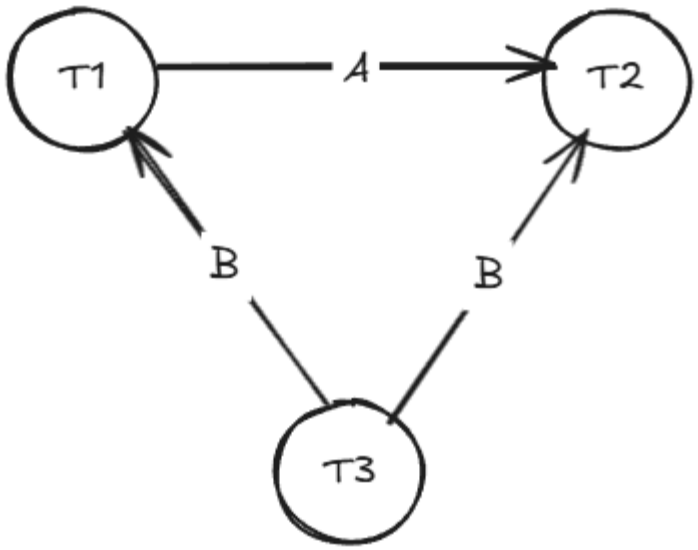
H1 : wl3(B); rl3(B); rl1(A); rl3(C); u3(B); rl1(B); u1(B); wl2(B); u3(C); rl2(C); u1(A); wl2(A); u2(B); u2(A); rl1(C); rl3(D); u3(D); u1(C); wl2(C); u2(C)

(a) Indicar a qu´e historia serial es equivalente H1. Justificar la respuesta.

(b) Responder: ¿Existe alg´un schedule legal no serializable que se pueda construir con T1, T2 y T3? Justificar la respuesta. En caso afirmativo, mostrar un ejemplo y justificar.

(c) Responder: ¿Es posible identificar en H1 un caso en el que pueda producirse un dirty read? Justificar la respuesta. En caso afirmativo, mostrar un ejemplo y justificar

T1	T2	T3
		wl3(B)
		rl3(B)
rl1(A)		
		rl3(C)
		u3(B)
rl1(B)		
u1(B)		
	wl2(B)	
		u3(C)
	rl2(C)	
u1(A)		
	wl2(A)	
	u2(B)	
	u2(A)	
rl1(C)		
		rl3(D)
		u3(D)
u1(C)		
	wl2(C)	
	u2(C)	



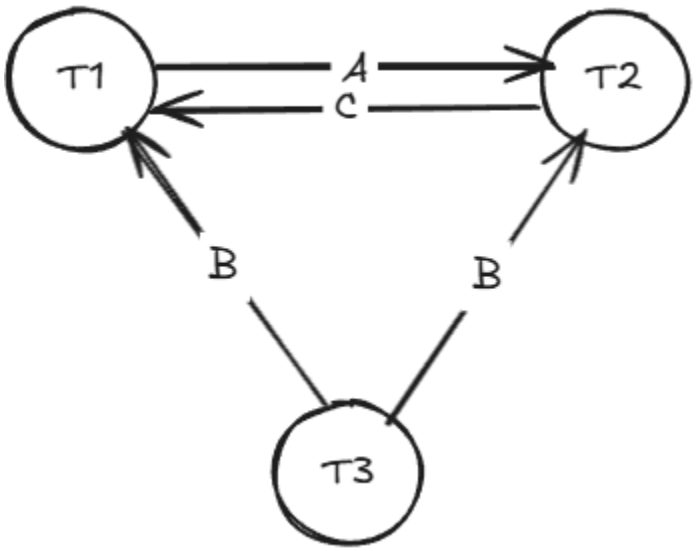
a) está bien??

El grafo no tiene ciclos. La historia serial equivalente es T3, T1, T2.

b)

T1	T2	T3
		wl3(B)
		rl3(B)
rl1(A)		
		rl3(C)
		u3(B)
rl1(B)		
u1(B)		
	wl2(B)	
		u3(C)
	rl2(C)	
u1(A)		
	wl2(A)	
	u2(B)	

	u2(A)	
		rl3(D)
		u3(D)
	wl2(C)	
	u2(C)	
rl1(C)		
u1(C)		



Cambiando el orden de estas operaciones sigue siendo una historia legal, pero se tiene que agregar una flecha de T2 a T1, se forma un ciclo T1, T2, no es serializable.

c) Sí, por ejemplo, cuando T3 hace wl3(B) y luego T1 hace rl1(B).

3.6. Dada la siguiente historia H1, sobre el conjunto de transacciones T1, T2, T3, T4 y el conjunto de items X, Y :

H1 = rl3(X); rl2(X); wl3(Y); u3(X); wl2(X); u3(Y); rl4(Y); u2(X); rl1(Y); rl4(X); u1(Y); u4(X); wl1(X); u4(Y); u1(X); c3; c2; c4; c1

Nota: Asuma que entre un wl y su correspondiente ul ocurre solamente una escritura del ´ıtem (no ocurre una lectura)

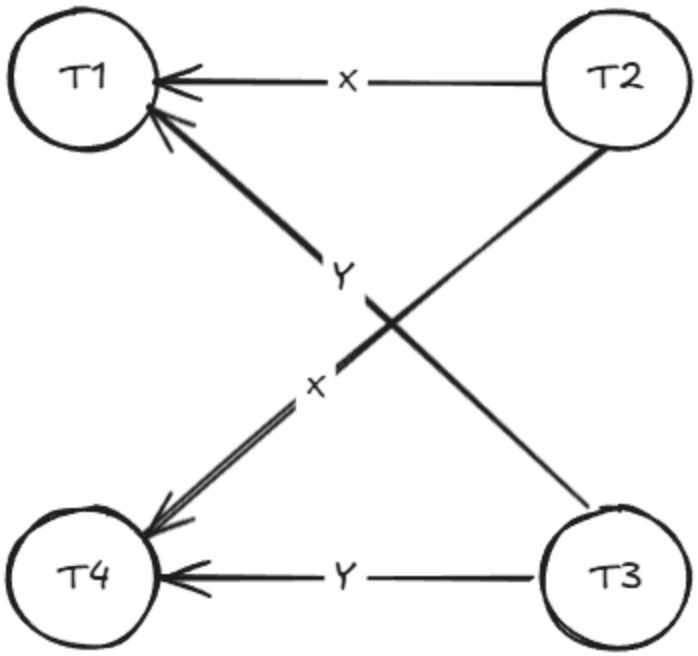
(a) Responder: ¿H1 es Legal? ¿Todas la Ti son 2PL? .

Es legal. Separo en transacciones para ver si son 2PL:

rl1(Y); u1(Y); wl1(X); u1(X); No es 2PL
rl2(X); wl2(X); u2(X); Es 2PL
rl3(X); wl3(Y); u3(X); u3(Y); Es 2PL
rl4(Y); rl4(X); u4(X); u4(Y) Es 2PL

(b) Hacer el SG(H1) e indicar si es serializable. En caso afirmativo obtener todas las historias seriales equivalentes.

T1	T2	T3	T4
		rl3(X)	
	rl2(X)		
		wl3(Y)	
		u3(X)	
	wl2(X)		
		u3(Y)	
			rl4(Y)
	u2(X)		
rl1(Y)			
			rl4(X)
u1(Y)			
			u4(X)
wl1(X)			
			u4(Y)
u1(X)			



Historias seriales equivalentes:
T2 o T3, T4 o T1

(c) Clasificar H1 con respecto a recuperabilidad: ST, ACA, RC, (no RC). Justificar

Es RC por el orden de los commits: c3; c2; c4; c1. Porque las únicas escrituras que después lee otra transacción son la de T3 que escribe primero antes de cualquier lectura de Y y después escribe T2 antes que cualquier lectura de X.

(d) Idem anterior pero cambiando el orden de los commits por: c4; c1; c2; c3

No es RC on ese orden. Porque no cumple la definición, porque por ejemplo T4 lee X después que la escribió T2 y el orden de los commits tiene c4 < c2.

(e) Ubicar los commits y modificar H1 para que las transacciones sean 2PL Estrictas

2PL Estricto (2PLE o *S2PL*)

Una transacción cumple con **2PL Estricto** si es 2PL y no libera ninguno de sus ***locks* de escritura** hasta **después** de realizar el *commit* o el *abort*. 2PLE garantiza que la historia es **ST**.

rl1(Y); wl1(X); u1(Y); u1(X);
rl2(X); wl2(X); u2(X);
rl3(X); wl3(Y); u3(X); u3(Y);

rl4(Y); rl4(X); u4(X); u4(Y)

rl3(X); rl2(X); **wl3(Y);** u3(X); **wl2(X); c3;** u3(Y); rl4(Y); **c2;** u2(X); rl1(Y); rl4(X); u4(X); **wl1(X);** u1(Y); u4(Y); c4; c1; u1(X);

3.7. Dadas las siguientes transacciones:

T1 = rl1(B); u1(B); wl1(A); u1(A)
T2 = rl2(A); wl2(A); u2(A)
T3 = rl3(A); rl3(B); u3(A); u3(B)

Nota: Asumir que entre un wl y su correspondiente ul ocurre solamente una escritura del ítem (no ocurre una lectura)

(a) Se pide hallar una historia H tal que sea: Legal; No Serial; Serializable; Equivalente a la ejecución serial T3, T2, T1 y ACA.

rl3(A); rl3(B); u3(A); rl2(A); **wl2(A);** u3(B); **c3;** u2(A); rl1(B); u1(B); **wl1(A)**; **c2;**u1(A); **c1**

(b) Hacer el SG(H) para verificar la seriabilidad del H hallado. Justificar las respuestas con respecto a legalidad y recuperabilidad.

Es legal no serial. Es ACA porque al leer siempre hubo un commit antes de la transacción que la escribió. No es ST porque tenemos wl2(A) < wl1(A) < c2, si fuera ST el c2 tendría que estar en el medio.

Es equivalente a T3, T2, T1 y a T2, T1, T3.

(c) Transformar las transacciones a 2PL Estricto y 2PL Riguroso

Estricto:
rl3(A); rl3(B); u3(A); rl2(A); **wl2(A);** u3(B); **c3;** u2(A); rl1(B); u1(B); **wl1(A)**; **c2;**u1(A); **c1**

Riguroso:
rl3(A); rl3(B); rl2(A); **c3;** u3(A); u3(B); **wl2(A); c2;** u2(A); rl1(B); ; **wl1(A)**; **c1;** u1(B) u1(A)

3.8. Dadas las siguientes transacciones:

T1 = rl1(A); r1(A); rl1(B); r1(B); wl1(B); w1(B); u1(A); u1(B)
T2 = rl2(A); r2(A); rl2(B); r2(B); u2(A); u2(B)

(a) Realizar un entrelazado serializable no serial

rl2(A); rl1(A); r1(A); r2(A); rl2(B); r2(B); u2(A); u2(B); rl1(B); r1(B); wl1(B); w1(B); u1(A); u1(B)

(b) Proponer un entrelazado donde un upgrade lock sea denegado y deba esperar.

rl2(A); r2(A); rl2(B); r2(B); rl1(A); r1(A); rl1(B); r1(B); wl1(B); w1(B); u2(A); u2(B); u1(A); u1(B)

en ese caso la transacción 1 debería esperar el unlock de la transacción 2 antes de escribir???

3.9. Tomando las siguientes transacciones

T1 = rl1(A); r1(A); wl1(A); w1(A); u1(A)
T2 = rl2(A); r2(A); wl2(A); w2(A); u2(A)

(a) Proponer un entrelazamiento que de un deadlock como resultado.

rl1(A); r1(A); rl2(A); r2(A); wl2(A); wl1(A); w2(A); u2(A); w1(A); u1(A)

T1 y T2 tiene rl pero piden wl y no se lo da porque ninguno hizo unlock, hay deadlock

(b) Reescribir las transacciones, pero ahora, usando un modelo que soporte update lock y verificar si puede resolverse el deadlock anterior.

T1 = ul1(A); r1(A); wl1(A); w1(A); u1(A)
T2 = ul2(A); r2(A); wl2(A); w2(A); u2(A)

ul1(A); r1(A); ul2(A); r2(A); wl2(A); wl1(A); w2(A); u2(A);w1(A); u1(A)
???????? no me quedó claro qué hay que hacer

3.10. Dadas las siguientes historias

H1 = r1(A); r2(B); r3(C); r1(B); r2(C); r3(D); w1(C); w2(D); w3(E)
H2 = r1(A); r2(B); r3(C); r1(B); r2(C); r3(D); w1(A); w2(B); w3(C)
H3 = r1(A); r2(B); r3(C); r1(B); r2(C); r3(A); w1(A); w2(B); w3(C)

Se pide:
(a) Insertar locks de lectura y escritura de tal forma que no haya upgrade locks. Explicar que ocurrir´ıa en la ejecuci´on con un planificador/scheduler que soporte locks de

lectura y de escritura
(b) Insertar locks de lectura y escritura de tal forma que si haya upgrade locks y describir que ocurre con la ejecuci´on.
(c) Insertar ahora locks de lectura, escritura y de actualizaci´on (update locks). Describir que ocurre en la ejecuci´on.