Repaso Equivalencia de Historias

Operaciones Conflictivas

Dos operaciones son conflictivas si operan sobre el mismo ítem y al menos **una** de ellas es una **escritura**.

Equivalencia

Dos historias H_i y H_j son conflicto equivalentes $H_i \equiv H_j$

- Están definidas sobre el mismo conjunto de transacciones
- El orden de las operaciones conflictivas de transacciones no abortadas es el mismo.

Repaso Testeo de Serializabilidad

Grafo de precedencia

Se utiliza el grafo de precedencia $SG(H)$. Es un grafo dirigido con las siguientes características:

- Un nodo para cada transacción $T_i \subseteq H$
- Hay ejes entre T_i y T_j sí y sólo sí hay una operación de T_i que precede en H a una operación de T_j y son operaciones conflictivas.
- Etiquetamos los ejes del grafo con los nombres de los ítems que los generan.

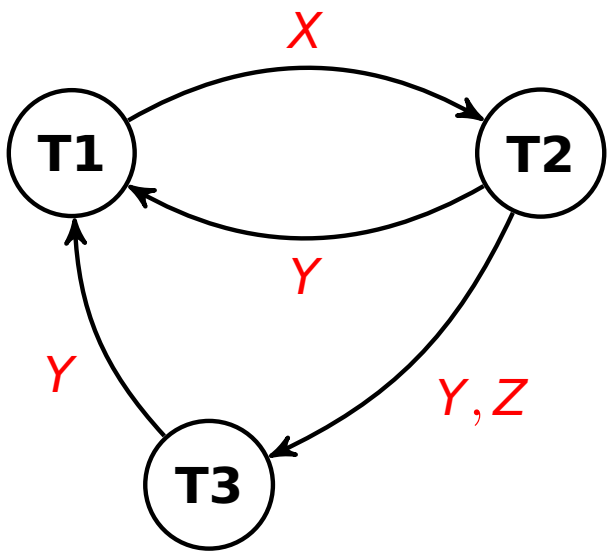
Teorema de la seriabilidad

Una historia H es **SR** sí y solo sí $SG(H)$ es acíclico.

Testeo de Serializabilidad - Ejemplo

- $H = r_2(Z); r_2(Y); w_2(Y); r_3(Y); r_3(Z); r_1(X); w_1(X); w_3(Y); w_3(Z); r_2(X); r_1(Y); w_1(Y); w_2(X)$

T_1	T_2	T_3
	$r_2(Z)$	
	$r_2(Y)$	
	$w_2(Y)$	
		$r_3(Y)$
		$r_3(Z)$
$r_1(X)$		
$w_1(X)$		
		$w_3(Y)$
		$w_3(Z)$
	$r_2(X)$	
$r_1(Y)$		
$w_1(Y)$		
	$w_2(X)$	



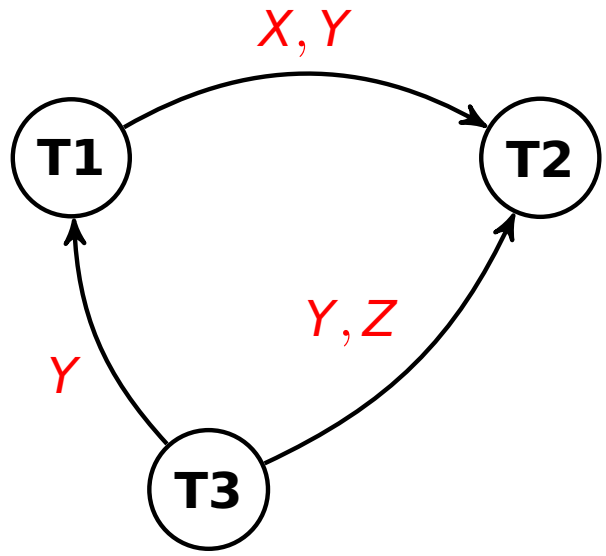
Tiene ciclos, ej:

$T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_2$

Testeo de Serializabilidad - Ejemplo 2

$H = r_3(Y); r_3(Z); r_1(X); w_1(X); w_3(Y); w_3(Z); r_2(Z); r_1(Y); w_1(Y); r_2(Y); w_2(Y); r_2(X); w_2(X)$

T_1	T_2	T_3
		$r_3(Y)$
		$r_3(Z)$
$r_1(X)$		
$w_1(X)$		
		$w_3(Y)$
		$w_3(Z)$
	$r_2(Z)$	
$r_1(Y)$		
$w_1(Y)$		
	$r_2(Y)$	
	$w_2(y)$	
	$r_2(X)$	
	$w_2(X)$	



Orden Serial:

Lectura entre transacciones

Dadas dos transacciones T_i y T_j decimos que T_i lee X de T_j si T_i lee X y T_j fue la última transacción que escribió X y no abortó antes de que T_i lo leyera.

1. $w_j(X) < r_i(X)$
2. $a_j \not< r_i(X)$
3. Si hay algún $w_k(X)$ tal que $w_j(X) < w_k(X) < r_i(X)$ entonces $a_k < r_i(X)$

Niveles de recuperabilidad

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*.

Teorema de la recuperabilidad

Teorema

$$ST \subset ACA \subset RC$$

Ortogonalidad

SR intersecta a todos los conjuntos RC, ACA y ST. Son conceptos ortogonales.

Es fácil ver que una historia serial es también ST.

Lock o Bloqueo Binario

Lock binario

El **lock binario** fuerza exclusión mutua sobre un ítem X.
Las transacciones pueden ser vistas como una secuencia de locks y unlocks

Historias Legales

Consistencia de Transacciones:

1. Una T_i puede leer o escribir un ítem X si previamente realizó un lock sobre X y no lo ha liberado
2. Si una transacción T_i realiza un lock sobre un elemento debe posteriormente liberarlo.

Legalidad:

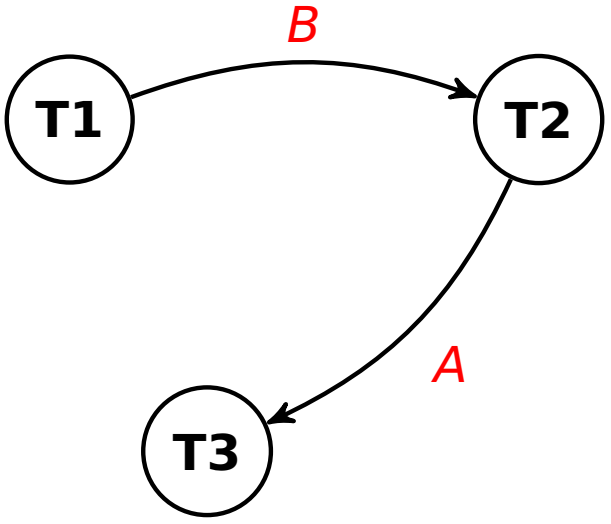
- Una T_i que desea obtener un lock sobre X que ha sido lockeado por T_j en un modo que conflictua, debe esperar hasta que T_j haga unlock de X.

Grafo de precedencia para lock binario

Se asume H legal. Para hacer el $SG(H)$ se siguen los siguientes pasos:

1. Hacer un nodo por cada $T_i \subseteq H$
2. Si T_i realiza un $l_i(X)$ para algún ítem X y luego T_j con $i \neq j$ realiza un $l_i(X)$ hacer un arco $T_i \rightarrow T_j$

Ejemplo: $H = l_2(A); u_2(A); l_3(A); u_3(A); l_1(B); u_1(B); l_2(B); u_2(B)$



H es SR y la Historia Serial Equivalente es T_1, T_2, T_3

Lock Ternario

Motivación

Debido a que operaciones de lectura de diferentes transacciones sobre el mismo ítem no son conflictivas se puede permitir que accedan sólo para lectura.

Atención

Sin embargo si una transacción desea escribir debe tener acceso exclusivo al ítem.

2 tipos de locks

$rl_i(A)$ **Lock de lectura o compartido**. La transacción i realiza un *bloqueo o lock* de lectura sobre el ítem A.
$wl_i(A)$ **Lock de escritura o exclusivo**. La transacción i realiza un *lock* exclusivo o de escritura sobre el ítem A.

Consistencia y Legalidad

Consistencia

(a) Una accion $r_i(X)$ debe ser precedida por un $rl_i(X)$ o un $wl_i(X)$, sin que intervenga un $u_i(X)$

(b) Un accion $w_i(X)$ debe ser precedida por una $wl_i(X)$ sin que intervenga un $u_i(X)$

(c) Todos los *locks* deben ser seguidos de un *unlock* del mismo elemento

Legalidad de las Historias

(a) Si $wl_i(X)$ aparece en una historia, entonces no puede haber luego un $wl_j(X)$ o $rl_j(X)$ para $j \neq i$ sin que haya primero un $u_i(X)$

(b) Si $rl_i(X)$ aparece en una historia no puede haber luego un $wl_j(X)$ para $j \neq i$ sin que haya primero un $u_i(X)$

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.

Conversión o Upgrading/Downgrading Lock

Conversión

Una transacción que tiene un *lock* sobre un ítem *X* tiene permitido bajo ciertas condiciones convertir dicho *lock* en otro tipo de *lock*.

La forma más común es el *upgrading lock*, es decir pasar de un *lock de lectura o compartido* **a un** *lock exclusivo o de escritura*.

Update Lock y Upgrade lock

Deadlock al usar updgrade lock

Supongamos T_1 y T_2, y se presenta la siguiente historia donde cada una quiere realizar un upgrade lock. Ambos son denegados:

$H = rl_1(X); rl_2(X); wl_1(X); wl_2(X)$[a]

[a] Las operaciones en rojo indican que no pudieron ser completadas y deben esperar

Update lock

Se puede evitar este problema del deadlock si agregamos otro modo de *lock* llamado ***update lock***. Un update lock sobre un ítem X que denotamos $ul_i(X)$ da a la transacción T_i privilegio de lectura sobre X pero no de escritura. Como ventaja el *update lock* pasa a ser el **único** que puede ser *upgraded* a *write lock*

Two Phase Locking - 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

Serializabilidad con 2PL

Dado $T = T_1, T_2, ..., T_n$, si toda T_i en T es **2PL**, entonces todA H legal sobre T es **SR**.

Variantes de 2PL

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*.

Serializabilidad con 2PL Estricto

2PLE garantiza que la historia es **ST**. No es libre de deadlock.

2PL Riguroso (2PLR)

Una transacción cumple con **2PL Riguroso** si es 2PL y no libera ninguno de sus ***locks* de escritura o lectura** hasta **después** de realizar el *commit* o el *abort*.
La serializabilidad es igual al orden de los *commit*

Detección usando Wait-for Graph

- Un nodo por cada transacción que tiene un lock o espera por uno.
- Un eje entre dos nodos (T_i y T_j) si T_i está esperando que T_j libere un lock que sobre un ítem que T_i necesita bloquear.

Considerar la siguiente historia parcial:
$l_1(A); l_2(B); l_1(B); l_3(C); l_2(C); l_4(B); l_3(A)$