Transaction Management

Concurrency Control (2)

Conflict Actions

- A pair of consecutive actions in a schedule constitutes a **conflict** if swapping these actions may change the effect of at least one of the transactions involved.
- Most pairs of actions do not cause a conflict.
- ri (X) and rj (Y) never cause a conflict, even if X = Y, since they do not modify the DB state.
- ri(X) and wj(Y) do not cause a conflict if $X \neq Y$.
- wi(X) and rj(Y) do not cause a conflict if $X \neq Y$.
- wi(X) and wj(Y) do not cause a conflict if $X \neq Y$.

Conflict Actions (cont.)

- The following three situations do cause a conflict:
- Actions of the same transaction, i.e. $i = j$.
- Two writes of the same database element by different transactions, i.e. wi(X) and wj(X), $i \neq j$. Depending on the schedule, the results of either wi(X) or wj(X) survive, which may be different.
- A read and a write of the same database element by different transactions, i.e. ri(X) and wj(X), $i \neq j$. ri(X) may read a different version of X.

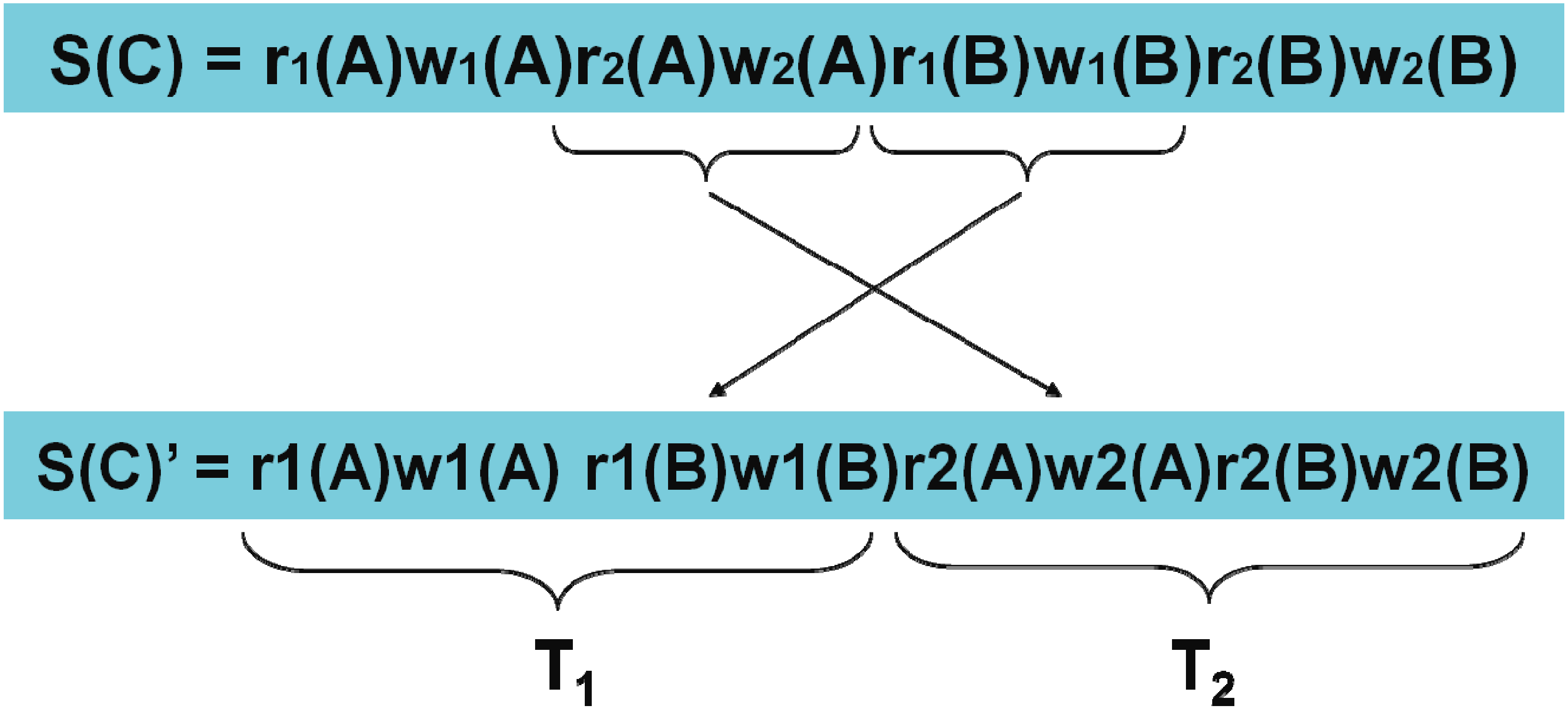
Conflict Equivalent/Serializable

- Definition:

S_1, S_2 are conflict equivalent schedules if S_1 can be transformed into S_2 by a series of swaps on non-conflicting actions.

A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

Review: Schedule C



$S(C) = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

S(C)' = r1(A)w1(A) r1(B)w1(B)r2(A)w2(A)r2(B)w2(B)

Conflict-Serializability

- If transactions Ti and Tj contain at least two pairs of conflicting actions, then for each of these pairs the action of Ti has to be performed before that of Tj (or always Tj before Ti).
- Given a schedule S, Ti *takes precendence over* Tj, denoted by Ti $<_S$ Tj, if there are actions pi(A) of Ti and qj(A) of Tj such that
  - pi(A) is ahead of qj(A) in S,
  - both pi(A) and qj(A) involve the same database element, and at least one of them is a write.

Conflict-Serializability

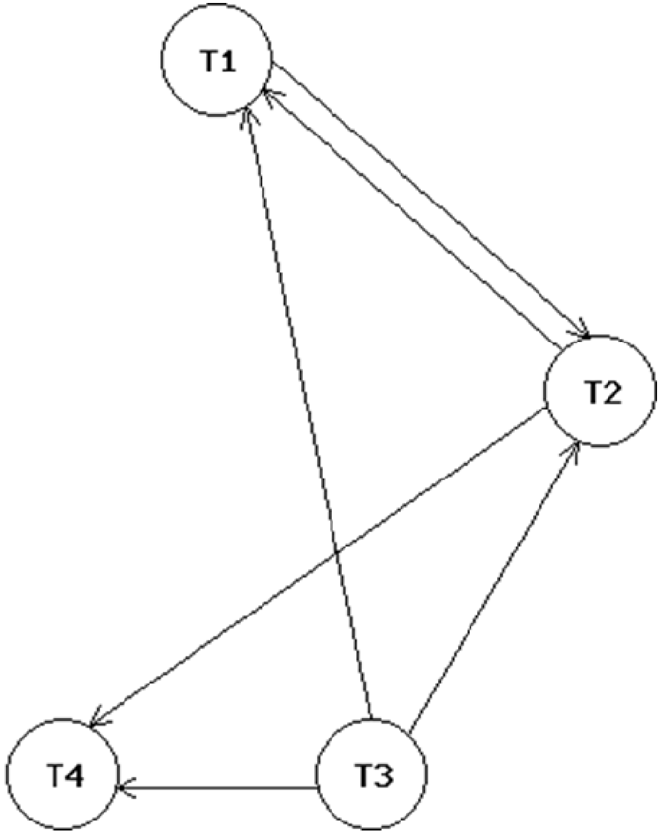
- If Ti takes precendence over Tj, then a schedule S' that is conflict equivalent to S must have pi(A) before qj(A).
- *Precedence graph*: directed graph with *nodes* representing the transactions of S,

  edges representing precedence relationships,

  i.e. edge from node Ti to Tj if Ti $<_S$ Tj.
- Notation: P(S)

Examples (1)

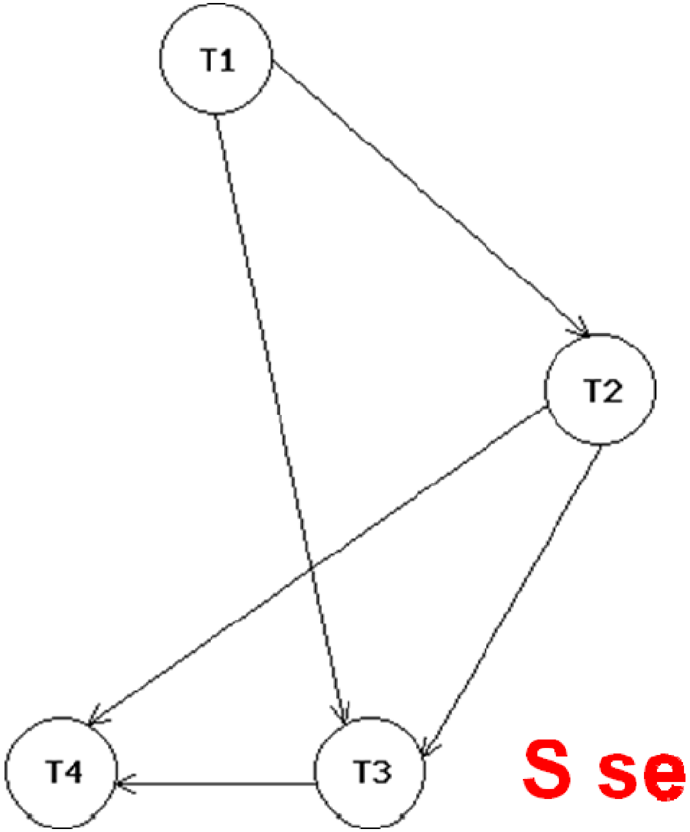
- What is P(S) for S=w3(A)w2(C)r1(A)w1(B)r1(C)w2(A)r4(A)w4(D)



Is S serializable?

Examples (2)

- What is P(S) for S=r1(A)w1(B)r1(C)w2(C)w2(A)w3(A)r4(A)w4(D)



S serial → P(S) acyclic!

Conflict-Serializability

- Lemma 1

  S1, S2 conflict equivalent ⇒ P(S1) = P(S2)

- Proof

  Assume P(S1) ≠ P(S2)

  ⇒ ∃ Ti, Tj: Ti → Tj in P(S1) and not in P(S2)

$$\Rightarrow \begin{array}{l} S1 = \ldots pi(A) \ldots\ qj(A) \ldots \\ S2 = \ldots qj(A) \ldots pi(A) \ldots \end{array} \left\{ \begin{array}{l} pi,\ qj \\ \text{in conflict} \end{array} \right.$$

  ⇒ S1, S2 not conflict equivalent

Conflict-Serializability

- Note

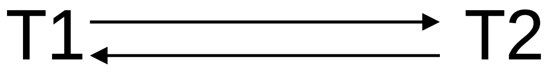
  P(S1)=P(S2) $\not\Rightarrow$ S1, S2 conflict equivalent

- Counter example

  S1=w1(A) r2(A) w2(B) r1(B)

  S2=r2(A) w1(A) r1(B) w2(B)

  P(S1)=P(S2)=



S1 not conflict equivalent to S2, since w1(A) and r2(A) cannot be swapped

Conflict-Serializability

- Theorem 2

  P(S) acyclic $\Longleftrightarrow$ S conflict serializable

- Proof

  (i) $\Leftarrow$

  Assume S is conflict serializable.

  $\Rightarrow$ $\exists$ S’: S’ is serial, S conflict equivalent to S’.

  $\Rightarrow$ P(S’) = P(S) according to Lemma 1.

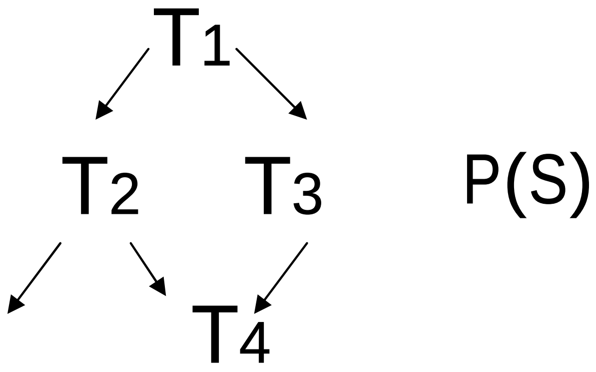
  P(S’) is acyclic because S’ is serial.

  $\Rightarrow$ P(S) is acyclic.

Conflict-Serializability

- Proof

(ii) ⇒



Assume P(S) is acyclic.

Transform S as follows:

(1) Take T1 to be transaction with no incoming edges.

T1 exists, since P(S) is acyclic.

(2) Move all T1 actions to the front:

S = ……. qj(A)…….p1(A)…..

This does not create any conflicts, since there is no Tj with Tj → T1.

(3) We now have S’ = < T1 actions ><... rest ...>.

(4) Repeat above steps to serialize rest.

Conflict-Serializability

- How to enforce that only conflict-serializable schedules are executed?
- There are two alternative approaches:

  \- *Pessimistic concurrency control*

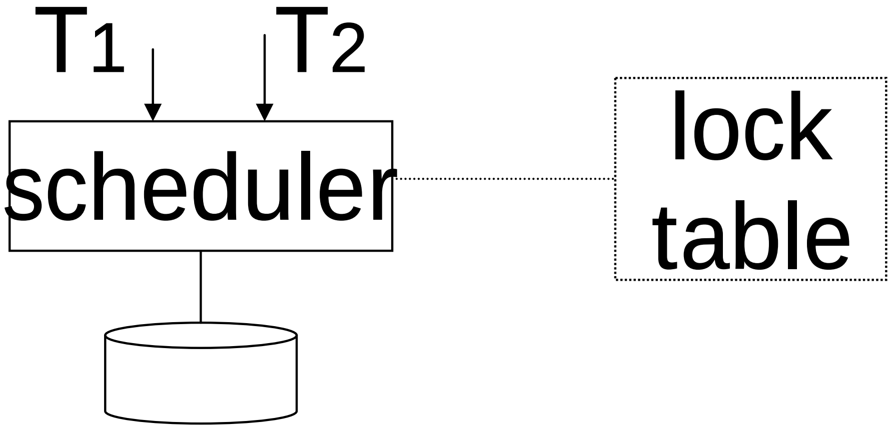
  Lock data elements to prevent P(S) cycles from occurring.

  \- *Optimistic concurrency control*

  Detect P(S) cycles and undo participating transactions, if necessary.

Enforcing Serializability by Locks

- Before accessing a database element, a transaction requests a *lock* on that element in order to prevent other transactions from accessing the same database element at the “same” time.
- Typically, different types of locks are used for different types of access operations, but we first introduce a simplified lock protocol with only one type of lock.



Enforcing Serializability by Locks

- We introduce two new actions:

  l_i (X): *lock* database element X
  u_i (X): *unlock* database element X, i.e. release lock.

- A locking protocol must guarantee the *consistency of transactions*:

  - A transaction can only read or write database X element if it currently holds a lock on X.

  - A transaction must unlock all database elements that is has locked at some later time.

- A consistent transaction is also called *well-formed*.

T_i: ... $l_i(A)$ ... $p_i(A)$ ... $u_i(A)$ ...

Enforcing Serializability by Locks

- A locking protocol must also guarantee the *legality of schedules*:
  At most one transaction can hold a lock on database element X at a given point of time.
- If there are actions l_i (X) followed by l_j (X) in some schedule, then there must be an action u_i(X) somewhere between these two actions.

$$S = \ldots\ldots\ldots\ li(A)\ \underset{\text{no } lj(A)}{\underleftrightarrow{\ldots\ldots\ldots\ldots\ldots}}\ ui(A)\ \ldots\ldots\ldots$$

Enforcing Serializability by Locks

- Example

S1 = l1(A)l1(B)r1(A)w1(B)l2(B)u1(A)u1(B)
r2(B)w2(B)u2(B)l3(B)r3(B)u3(B)

→ S1 illegal, because T2 locks B before T1 has unlocked it

S2 = l1(A)r1(A)w1(B)u1(A)u1(B)
l2(B)r2(B)w2(B)l3(B)r3(B)u3(B)

→ T1 inconsistent, because T1 writes B before locking it

S3 = l1(A)r1(A)u1(A)l1(B)w1(B)u1(B)
l2(B)r2(B)w2(B)u2(B)l3(B)r3(B)u3(B)

→ schedule legal and all transactions consistent

To-Do-List

- Do a research on how the currency control and logging recovery are related.