1. Transaction Processing
   1. Consider schedules S1 and S2 below.

S1: r3(X), r3(Z), r1(Y), r1(X), w3(Z), r2(Y), w1(X), r2(X), w2(X), r1(Z), w1(Z), r2(Z), w2(Y

S2: r2(X), r3(Y), r2(Z), w3(Y), r1(X), w2(Z), r1(Y), r3(Z), r1(Z), r2(Y), w2(Y), w3(Z), w1(X), w1(Z)

* 1. Apply the basic timestamp ordering (BTO) algorithm to schedules S1 and S2. Determine whether or not the algorithm allows the execution of the schedules, and discuss cascading rollback (if any).

Hints: each operation takes one time unit, and timestamp of each transaction is the time associated to its first operation. For example, timestamps of transactions T1, T2, and T3 in schedule S1 are 3, 6, and 1 (respectively).

[20 marks]

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | T1 | T2 | T3 |
| r3(X) | - | - | 1 |
| r3(Z) | - | - | 1 |
| r1(Y) | 3 | - | - |
| r1(X) | 3 | - | - |
| w3(Z) | - | - | 1 |
| r2(Y) | - | 6 | - |
| w1(X) | 3 | - | - |
| r2(X) | - | 6 | - |
| w2(X) | - | 6 | - |
| r1(Z) | 3 | - | - |
| w1(Z) | 3 | - | - |
| r2(Z) | - | 6 | - |
| w2(Y) | - | 6 | - |

* + 1. *Assign timestamps to each transaction. The timestamp of a transaction is the time associated with its first operation. For example, in schedule S1, the timestamps of transactions T1, T2, and T3 are 3, 6, and 1 respectively.*
    2. *Execute the transactions in timestamp order. If a transaction tries to execute out of order, roll it back and restart it.*
    3. *We keep track of the read and write timestamps for each data item. The read timestamp (RTS) is the timestamp of the most recent transaction that read the item, and the write timestamp (WTS) is the timestamp of the most recent transaction that wrote the item.*
    4. *When a transaction T wants to read an item X, check the WTS of X. If the WTS is greater than the timestamp of T, then T is trying to read a value of X that is written by a future transaction, so roll back and restart T.*
    5. *When a transaction T wants to write an item X, check both the RTS and WTS of X. If either is greater than the timestamp of T, then T is trying to write a value of X that has been read or written by a future transaction, so roll back and restart T.*
    6. *If T can read or write X without violating the above conditions, update the RTS or WTS of X and continue.*

*We repeat steps 4-6 until all transactions are executed.*

* + 1. *A cascading rollback occurs when the rollback of one transaction forces the rollback of other transactions that have read the data written by the rolled back transaction.*
    2. *Now, let's apply these steps to schedules S1 and S2. The specific steps and results would depend on the exact operations and data items in the schedules. For schedule S1: S1: r3(X), r3(Z), r1(Y), r1(X), w3(Z), r2(Y), w1(X), r2(X), w2(X), r1(Z), w1(Z), r2(Z), w2(Y) The timestamps of transactions T1, T2, and T3 are 3, 6, and 1 respectively.*
    3. *Start with the first operation r3(X). Since T3's timestamp is 1, it can read X. Set RTS(X) = 1.*
    4. *The next operation is r3(Z). T3 can read Z. Set RTS(Z) = 1.*
    5. *The next operation is r1(Y). Since T1's timestamp is 3, it can read Y. Set RTS(Y) = 3.*
    6. *The next operation is r1(X). T1 can read X since RTS(X) = 1 < 3. Update RTS(X) = 3.*
    7. *The next operation is w3(Z). T3 can write Z since WTS(Z) is not set and RTS(Z) = 1 <= 1. Set WTS(Z) = 1.*
    8. *We continue this process for the rest of the operations. If a transaction tries to read or write a data item that has been written by a future transaction, roll it back and restart it.*

1. Testing the serializability of S1 and S2 by serialization graph technique to prove that the successful execution of a schedule under BTO will ensure the serializability of the schedule.

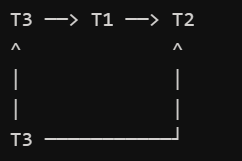
[15 marks]

* 1. *For each pair of transactions Ti and Tj where Ti executes before Tj, draw an edge from Ti to Tj.*
  2. *Check for cycles in the graph. If there are no cycles, the schedule is serializable.*
  3. *An arrow from Ti to Tj indicates that transaction Ti must execute before transaction Tj due to a read-write or write-write conflict.*

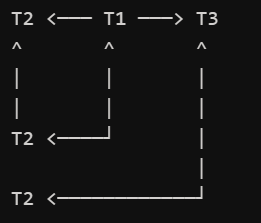
*For S1, there are no cycles, which suggests that S1 is serializable.*

*For S2, there are also no cycles, indicating that S2 is serializable as well.*

* 1. *Let's apply these steps to schedules S1 and S2:* 
     1. *For S1: r3(X), r3(Z), r1(Y), r1(X), w3(Z), r2(Y), w1(X), r2(X), w2(X), r1(Z), w1(Z), r2(Z), w2(Y)*
     2. *Based on the execution order from 3(a), we can draw the edges for each pair of transactions that have a read-write or write-write conflict. For example, if T3 reads X before T1 writes X, draw an edge from T3 to T1.*
     3. *We repeat this process for all pairs of transactions in S1. Then, check for cycles in the graph. If there are no cycles, S1 is serializable.*

**

* 1. *For S2: r2(X), r3(Y), r2(Z), w3(Y), r1(X), w2(Z), r1(Y), r3(Z), r1(Z), r2(Y), w2(Y), w3(Z), w1(X), w1(Z) we repeat the same process for S2.*

**

* 1. *Draw the edges for each pair of transactions that have a read-write or write-write conflict, then check for cycles in the graph.*
  2. *Serialization graph for S1:*
     + *T1 -> ['T2', 'T2', 'T2', 'T2', 'T2', 'T2', 'T2', 'T2']*
     + *T2 -> []*
     + *T3 -> ['T1', 'T1', 'T2', 'T2', 'T1', 'T1', 'T2', 'T1', 'T1', 'T2']*
  3. *Serialization graph for S2:*
     + *T1 -> ['T2', 'T2', 'T3']*
     + *T2 -> ['T1', 'T1', 'T3', 'T1', 'T3', 'T1', 'T3', 'T1', 'T3', 'T1']*
     + *T3 -> ['T1', 'T2', 'T2', 'T1', 'T2', 'T2', 'T1', 'T1', 'T1']*
  4. *If there are no cycles, S2 is serializable. Remember, this is a manual process and requires careful tracking of the operations in each transaction and the order in which they execute. It's also important to understand the rules of the serialization graph technique and apply them correctly.*
  5. *Now, let's go into more detail about the serialization graph technique: The serialization graph is a directed graph where each node represents a transaction and each edge represents a conflict between transactions.*
  6. *A conflict occurs when one transaction reads or writes a data item that another transaction later writes. There are two types of conflicts: read-write (RW) and write-write (WW).*
  7. *A RW conflict will occurs when a transaction reads a data item that another transaction later writes, and a WW conflict occurs when a transaction writes a data item that another transaction later writes. When constructing the serialization graph, we draw an edge from Ti to Tj for each RW or WW conflict where Ti executes before Tj.*
  8. *This indicates that Ti must execute before Tj to maintain the same order of operations on the conflicting data item. Once the serialization graph is constructed, we check for cycles. A cycle in the serialization graph indicates a circular dependency between transactions, which means that the schedule is not serializable. If there are no cycles, the schedule is serializable, which means that it is equivalent to some serial schedule of the transactions. This process ensures that the successful execution of a schedule under the Basic Timestamp Ordering (BTO) algorithm will ensure the serializability of the schedule, as required by question 3(b).*

1. Deductive Database

REACHABLE (X, Y): - CITY(X), CITY(Y), FLIGHT (X, Y)

REACHABLE (X, Y): - CITY(X), CITY(Z), FLIGHT (X, Z), REACHABLE (Z, Y)

Where REACHABLE (X, Y) means that city Y can be reached from city X, and FLIGHT (X, Y) means that there is a flight from city X to city Y (Note: No flight in reverse direction can be automatically assumed).

1. Construct fact predicates that describe the following:

New Delhi (del), Beijing (pek), Saigon (sgn), Auckland (akl), Singapore (sin) and Brisbane (bne) are cities.

The following 5 flights exist: sin to del, del to pek, pek to sgn, akl to sin and bne to akl (Note: No flight in reverse direction can be automatically assumed). [5 marks]

Construct a model theoretic interpretation (that is, an interpretation similar to the one shown in Figure 24.13, Lecture Notes) of the above rules using the given facts. [5 marks]

* + *Is there a guarantee of reachability between any 2 cities? Give reason(s) or example(s) to support your answer. [5 marks]*
    1. *% Define the cities*

*city(del).*

*city(pek).*

*city(sgn).*

*city(akl).*

*city(sin).*

*city(bne).*

* + 1. *% Define the flights*

*flight(sin, del).*

*flight(del, pek).*

*flight(pek, sgn).*

*flight(akl, sin).*

*flight(bne, akl).*

* + *Based on the facts and rules, there isn't a guarantee of reachability between any two cities. The reachability depends on the existence of direct flights or a sequence of flights connecting the cities. For example, according to the facts provided, there is a direct flight from Singapore (sin) to New Delhi (del), so New Delhi is reachable from Singapore. Similarly, there is a sequence of flights from Singapore to Beijing (pek) via New Delhi, so Beijing is reachable from Singapore. However, there is no flight from New Delhi (del) to Brisbane (bne) either directly or through a sequence of flights. Therefore, Brisbane is not reachable from New Delhi. The reachability between any two cities can be determined by querying the REACHABLE (X, Y) predicate in the Prolog program. If the query returns true, then city Y is reachable from city X. If the query returns false, then city Y is not reachable from city X.*

1. There is 1 new flight added: sgn to bne.

Update the model theoretic interpretation to include the new flight. [5 marks]

Prove that REACHABLE (pek, akl) is true. Show your work at each step. [5 marks]

Is there a guarantee of reachability between any 2 cities? Give reason(s) or example(s) to support your answer.

[5 marks]

* + *% Add the new flight*

*flight(sgn, bne).*

* + *We prove that REACHABLE (pek, akl) is true. This means that there is a sequence of flights from Beijing (pek) to Auckland (akl). We can use the REACHABLE rule to find a sequence of flights. The REACHABLE rule states that city Y is reachable from city X if there is a direct flight from X to Y, or if there is a city Z such that there is a flight from X to Z and city Y is reachable from city Z. Let's apply this rule to prove that REACHABLE(pek, akl) is true:*
  + *There is a direct flight from Beijing (pek) to Saigon (sgn). So, Saigon is reachable from Beijing.*
  + *There is a direct flight from Saigon (sgn) to Brisbane (bne). So, Brisbane is reachable from Saigon, and therefore, Brisbane is reachable from Beijing.*
  + *There is a direct flight from Brisbane (bne) to Auckland (akl). So, Auckland is reachable from Brisbane, and therefore, Auckland is reachable from Beijing.*
  + *So, we have shown that REACHABLE(pek, akl) is true. with the addition of the new flight from Saigon to Brisbane, all cities are now reachable from any other city. For example, even though there is no direct flight from New Delhi (del) to Brisbane (bne), Brisbane is now reachable from New Delhi through a sequence of flights: New Delhi to Beijing, Beijing to Saigon, and Saigon to Brisbane*

1. The following predicates are added:

DURATION(del, pek, 6).

DURATION(sin, del, 5.5).

DURATION(sgn, bne, 8).

DURATION(pek, sgn, 5).

DURATION(bne, akl, 3).

DURATION(akl, sin, 10.5).

Note: DURATION(X, Y, Z) means that a flight from X to Y is in Z hours. 5.5 means 5 hours and 30 minutes.

Given a rule named as REACHABLE\_AND\_DURATION(X, Y, Z):

REACHABLE\_AND\_DURATION(X, Y, Z) :- FLIGHT(X, Y), DURATION(X, Y, Z

REACHABLE\_AND\_DURATION(X, Y, Z) :- FLIGHT(X, T), DURATION(X, T, K), REACHABLE\_AND\_DURATION(T, Y, H), K+H = Z

The rule means that city Y can be reached from city X and the total hours of flights is Z hours. Assume that we have a built-in comparison predicate “=” which allows us to check equality between 2 arguments. And a built-in arithmetic function “+” that allows us to sum 2 numeric arguments.

Prove that REACHABLE\_AND\_DURATION(bne, del, 19) is true. Show your work at each step. [10 marks]

Consider the following query: What cities are reachable in less than 20 hrs of flights from Delhi? Write a new rule and write a query based on that rule (you do not need to prove the query). Assume that we have a built-in comparison predicate “<” which allows us to check inequality between 2 arguments. [10 marks]

* + *% Adding the duration predicates*

*duration(del, pek, 6).*

*duration(sin, del, 5.5).*

*duration(sgn, bne, 8).*

*duration(pek, sgn, 5).*

*duration(bne, akl, 3).*

*duration(akl, sin, 10.5).*

* + *We can use the REACHABLE\_AND\_DURATION rule to find a sequence of flights. The REACHABLE\_AND\_DURATION rule states that city Y is reachable from city X with a total duration of Z hours if there is a direct flight from X to Y with a duration of Z hours, or if there is a city T such that there is a flight from X to T with a duration of K hours and city Y is reachable from city T with a total duration of H hours, and K + H equals Z. Applying this rule to prove that REACHABLE\_AND\_DURATION(bne, del, 19) is true:*
  + *There is a direct flight from Brisbane (bne) to Auckland (akl) with a duration of 3 hours. So, Auckland is reachable from Brisbane with a total duration of 3 hours.*
  + *There is a direct flight from Auckland (akl) to Singapore (sin) with a duration of 10.5 hours. So, Singapore is reachable from Auckland with a total duration of 10.5 hours, and therefore, Singapore is reachable from Brisbane with a total duration of 3 + 10.5 = 13.5 hours.*
  + *There is a direct flight from Singapore (sin) to New Delhi (del) with a duration of 5.5 hours. So, New Delhi is reachable from Singapore with a total duration of 5.5 hours, and therefore, New Delhi is reachable from Brisbane with a total duration of 13.5 + 5.5 = 19 hours.*
  + *So, we have shown that REACHABLE\_AND\_DURATION(bne, del, 19) is true. we can write a new rule REACHABLE\_IN\_LESS\_THAN\_20\_HRS(X, Y) and a query based on that rule. The REACHABLE\_IN\_LESS\_THAN\_20\_HRS(X, Y) rule states that city Y is reachable from city X in less than 20 hours if there is a city Z such that city Y is reachable from city X with a total duration of Z hours and Z is less than 20*
  + *% Define the new rule*

*reachable\_in\_less\_than\_20\_hrs(X, Y) :- reachable\_and\_duration(X, Y, Z), Z < 20.*

*% Query the new rule*

*?- reachable\_in\_less\_than\_20\_hrs(del, Y).*