1 The Transaction Concept

A **transaction** is a sequence of operations that should be treated as a single operation, e.g. a bank transfer from A to B:

```
T:  read(A)         read(A)   <- we only need to look at
    A -= 100        write(A)     reads and writes
    write(A)        read(B)
    read(B)         write(B)
    B += 100
    write(B)
```

We only analyze **reads** and **writes**.

Transactions must have the following properties: Atomicity, Consistency, Isolation, and Durability.

2 Atomicity

Atomicity is the requirement that **all or none** of the instructions of a transactions should happen. This is implemented with **logs**, allowing the DBS to **roll back** to a consistent state in the event of a transaction failing. The following state transition diagram describes the life of a transaction:

3 Consistency

Consistency is the requirement that if the DB is consistent before the transaction, it must be consistent after the transaction. This can be enforced by **integrity constraints**, but is more or less up to the DB designer as he defines the semantics of the DB.

4 Isolation

Isolation is the requirement that concurrent (interleaved) execution of the instructions of multiple transactions results in a state equivalent to a serial execution. The **concurrency control** component of the DBS handles isolation.

4.1 Concurrency

We execute transactions **concurrently** to improve throughput by way of pipelining, and to utilize resources such as disk I/O effeciently.

A **schedule** is a execution sequence for the instructions of 2 or more transactions. If each transaction completes before any of the others start, the schedule is **serial**. A schedule is **correct** if it is

result equivalent to a serial schedule.

4.2 Conflict Serializability

Conflict serializability is a strong notion of serializability. It is defined as

conflict equivalence to a serial schedule.

Neighbouring instructions on the same data item in different transactions **conflict** if one or more of them is a **write**. Non-conflicting neighbouring instructions can be **swapped** to produce a conflict **equivalent schedule**.

A write to a data item without a prior read is called a **blind write**. Such an instruction always conflict and cannot be swapped.

Conflict serializability can be tested by creating a directed **precedence graph**, where **nodes represent transactions** and **edges represent conflicts**. If a precedence graph for a schedule contains a **cycle**, the schedule is **conflict serializable**.

4.3 View Serializability

View serializability is a weaker notion of serializability, meaning that all conflict serializable schedules are also view serializable. It is defined as

view equivalence to a serial schedule.

View equivalence between 2 schedules is defined as:

For all data items Q:

- **first reader** - if in one schedule T1 reads the initial value of Q in one schedule, it must also do so in the other.
- **write before read** - if in one schedule T1 writes to Q, and T2 then reads Q, this sequence must be kept in the other schedule
- **last writer** - if in one schedule T1 writes the final value of Q, it must also do in the other

View serializable schedules that are **not** conflict serializable have **blind writes**.

5 Durability

Durability is the requirement that when a transaction is succesfully **completed,** no system failure (except HD failure :) will result in data loss. This is implemented in the **recovery management** component of the DBS, updates are textbflogged to disk before being carried out, and can be replayed in the event of system failure. Also transactions are not committed before the update is flushed to disk.