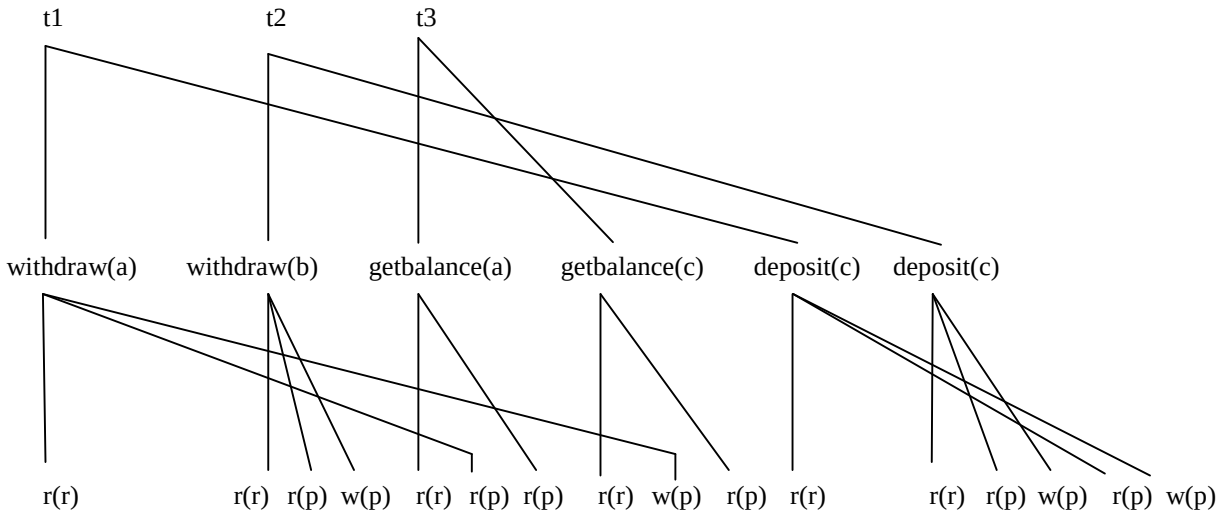
UNIVERSITÄT DES SAARLANDES
Fachrichtung 6.2 – Informatik
Prof. Dr.-Ing. G. Weikum
Dipl.-Inform. R. Schenkel

4. Übung zur Vorlesung *Transaktionsorientierte Informationssysteme* im Sommersemester 2000

4.1. Consider the following 2-level schedule consisting of two funds-transfer transactions and a balance-lookup transaction. Assume that the getbalance operations do not commute with withdraw operations nor with deposits. Is this schedule tree-reducible? If so, which is the equivalent serial execution? If not, how would the schedule have to changed (as little as possible) so as to render it tree-reducible?



4.2. Consider a data server with a SQL-style interface (see Chapter 1) and a database with a *Person* table whose rows contain a unique *Name* attribute and a *City* attribute. The server executes operations like

- `select * from Person where City=c`, where `c` is a parameter, for looking up all persons that live in a given city, and
- `update Person set City=c where Name=n`, for recording that a given person has moved to a new city.

Let us abbreviate these two operation types as *Select(c)* and *Update(n,c),* respectively. The server executes these operations by decomposing them into index lookups (*search(key)*), record fetches (*fetch(rid)*), record modifications (*modify(rid)*), and index maintenance steps (*insert(key,rid)* and *delete(key,rid)*). Assume that B^+-tree index structures exist for each of the two relevant attributes, and that both of these trees have depth two, i.e., consist of a root node and a leaf level. All of the mentioned record and index operations are finally transformed into page reads and writes.

Now consider a transaction that finds all persons from two different cities, say Los Angeles and New York, and a second transaction that records the move of a couple, say Jerry and Liz Smith, from Los Angeles to New York. Model the executions of both of these transactions as 3-level transactions, and discuss possible schedules for them. Give (non-trivial, e.g., non-serial) examples for 3-level schedules that are a) tree-reducible and b) not tree-reducible and thus not admissible.

4.3. Consider again the example of Figure 6.6. Give the necessary transformation steps, using the commutativity and the tree pruning rules, for proving that the schedule is tree-reducible.

4.4. Consider a counter object with operations with lower and upper bounds as constraints. The counter interface supports three operations: *Increment* adds a specified value $\triangle$ to the counter and returns "ok" if the upper bound is not exceeded; otherwise it returns "no" and leaves the counter unchanged. Analogously, *decrement* subtracts a given value from the counter or returns "no" if the lower bound became violated. Finally, *getvalue* returns the counter value. Give a return-value commutativity table for these three operations. Discuss possible improvements of concurrency by taking into account explicit constraints among operation parameters.

4.5. Construct an example schedule with operations on FIFO queues that demonstrate the higher concurrency that return-value commutativity can achieve over general commutativity.

4.6. Construct the return-value commutativity table for a semiqueue. Show, by means of example schedules, that semiqueues allow higher concurrency than FIFO queues.

4.7. Consider the following execution of operations on an initially empty queue q where a, b, and c are entries added into the queue:

$$enq_1(q,a)\ enq_2(q,b)\ deq_3(q)\ enq_1(q,c)\ deq_3(q)$$

Discuss whether this schedule is serializable assuming a) general commutativity, b) return-value commutativity, and c) return-value commutativity for semi-queues with nondeterministic selection of entries by dequeue operations. In the last setting, which entries should be selected by the two dequeue operations of t_3 to produce an intuitively correct execution?

4.8. Design an abstract data type "mailbox" with operations like *send* (i.e., add a message to the mailbox), *receive* (i.e., extract a message from the mailbox), and so on. Discuss the operations' commutativity relation, for a) general (i.e., state-independent) commutativity and b) return-value commutativity. Devise an implementation of mailbox objects in terms of queues, records, and ultimately pages. Sketch the resulting transaction trees. Give an example for a concurrent execution, which as much concurrency as possible, for two transactions where one sends two messages to two different mailboxes and the other receives all messages from one of the two mailboxes.

4.9. Even if two operations do not commute in the strict sense, the different effects that result from the two possible orderings may be considered as "effectively equal" from an application viewpoint. One could then simply declare the two operations as *compatible* and treat them as if they were commutative. An example would be a deposit operation together with a new style of withdraw operation that allows overdrafting but claims a penalty of say $10 each time the account balance is below the specified threshold and money is withdrawn. So the "code" for this relaxed variant of withdraw would be

```
relaxed_withdraw(x,△):
      x.balance := x.balance - △;
      if x.balance < 0 then
            x.balance := x.balance - 10
      fi;
```

Give an example schedule with these operations that would be considered conflict-serializable relative to the compatibility of the operations, but is not conflict-serializable under a strict interpretation of commutativity. Discuss the acceptability of the resulting (worst-case) situation from an application viewpoint, of both the bank and the customer.