Figure 20.1 Lock and unlock operations for binary locks.

```
lock_item (X):

B: if LOCK (X)=0  (* item is unlocked *)
   then LOCK (X)←1  (* lock the item *)
   else begin
      wait  (until lock (X)=0  and
        the lock manager wakes up the transaction);
      go to B
      end;


unlock_item (X):

LOCK (X)←0;  (* unlock the item *)
if any transactions are waiting
  then wakeup one of the waiting transactions;
```

Figure 20.2 Locking and unlocking operations for two-mode (read-write or shared-exclusive) locks.

```
read_lock (X):
  B: if LOCK (X)="unlocked"
     then begin LOCK (X)← "read-locked";
                no_of_reads(X)← 1
          end
     else if LOCK(X)="read-locked"
          then no_of_reads(X)← no_of _reads(X) + 1
          else begin wait (until LOCK (X)="unlocked" and
                    the lock manager wakes up the transaction);
               go to B
               end;

write_lock (X):
  B: if LOCK (X)="unlocked"
     then LOCK (X)← "write-locked"
     else begin
          wait (until LOCK(X)="unlocked" and
            the lock manager wakes up the transaction);
          go to B
          end;

unlock_item (X):
  if LOCK (X)="write-locked"
     then begin LOCK (X)← "unlocked;"
          wakeup one of the waiting transactions, if any
          end
     else if LOCK(X)="read-locked"
          then begin
               no_of_reads(X)← no_of_reads(X) – 1;
               if no_of_reads(X)=0
                 then begin LOCK (X)="unlocked";
                      wakeup one of the waiting transactions, if any
                      end
                 end;
```

Figure 20.3 Transactions that do not obey two-phase locking. (a) Two transactions T_1 and T_2. (b) Results of possible serial schedules of T_1 and T_2. (c) A nonserializable schedule S that uses locks.

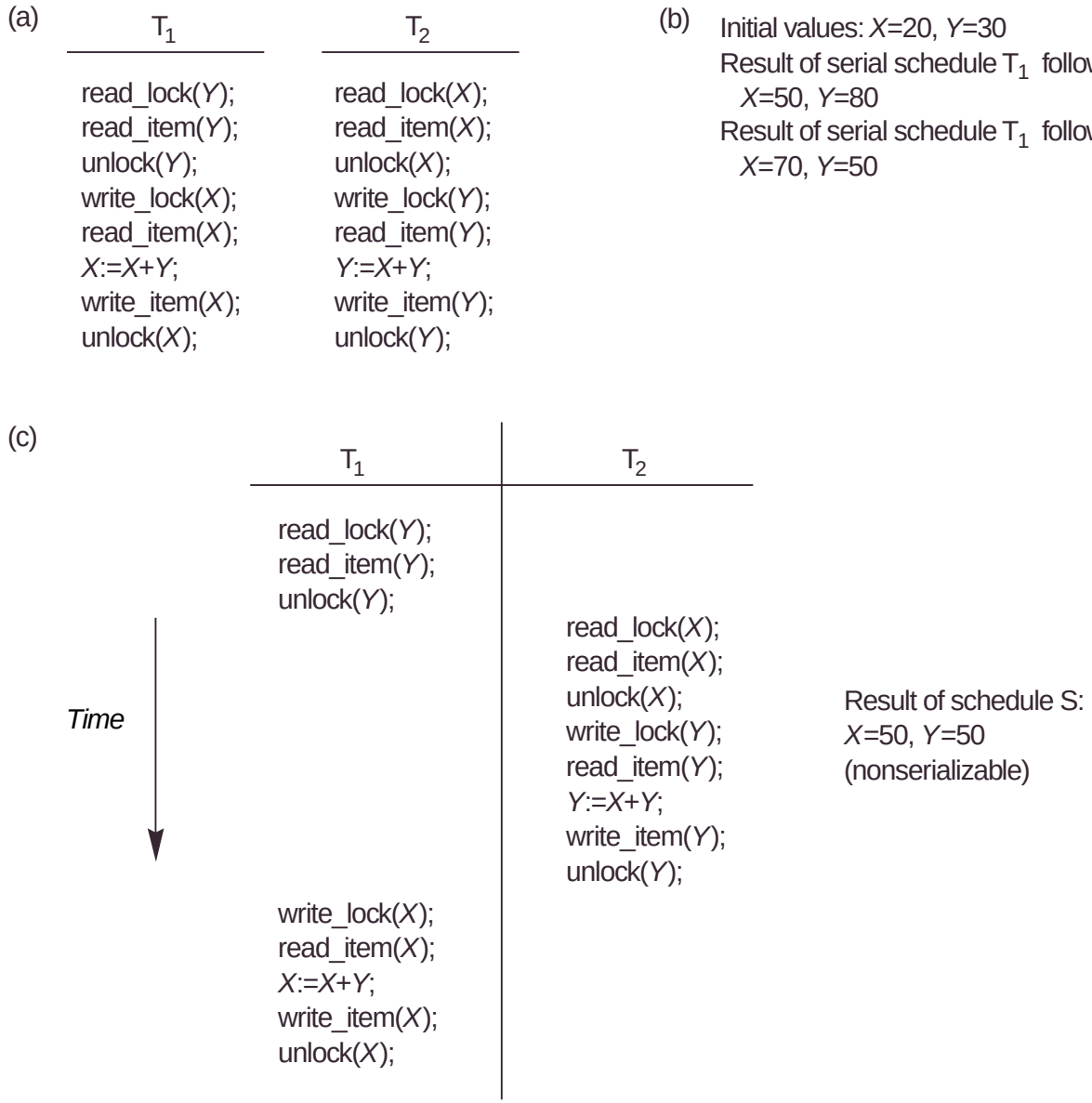

Figure 20.4 Transactions T_1' and T_2', which are the same as T_1 and T_2 of Figure 20.3 but which follow the two-phase locking protocol. Note that they can produce a deadlock.

T_1'	T_2'
read_lock (*Y*);	read_lock (*X*);
read_item (*Y*);	read_item (*X*);
write_lock (*X*);	write_lock (*Y*);
unlock (*Y*);	unlock (*X*);
read_item (*X*);	read_item (*Y*);
X:=*X*+*Y*;	*Y*:=*X*+*Y*;
write_item (*X*);	write_item (*Y*);
unlock (*X*);	unlock (*Y*);

Figure 20.5 Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).

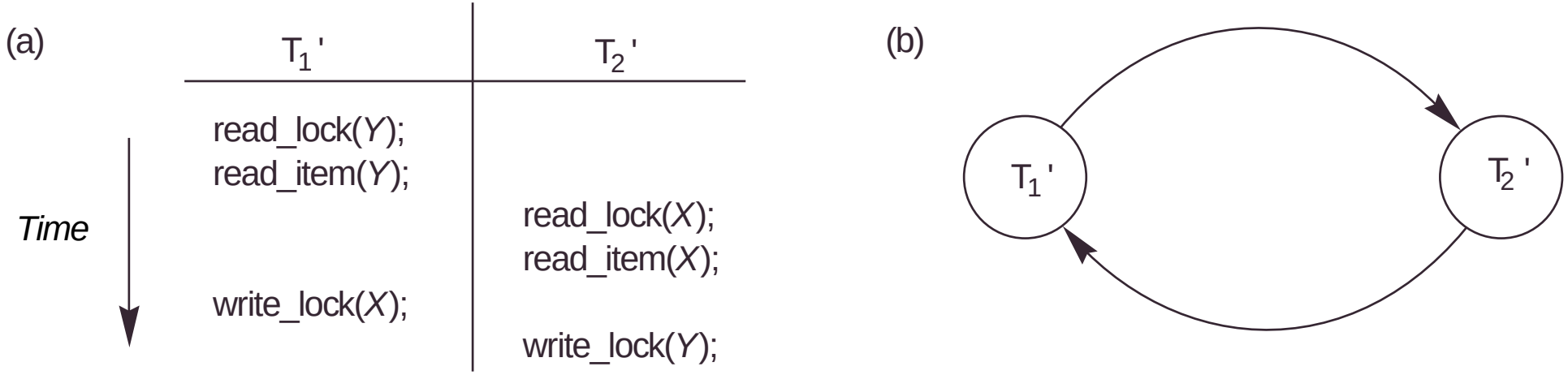

Figure 20.6 Lock compatibility tables. (a) A compatibility table for read/write locking scheme. (b) A compatibility table for read/write/certify locking scheme.

(a)

	Read	Write
Read	yes	no
Write	no	no

(b)

	Read	Write	Certify
Read	yes	yes	no
Write	yes	no	no
Certify	no	no	no

Figure 20.7 A granularity hierarchy for illustrating multiple granularity level locking.

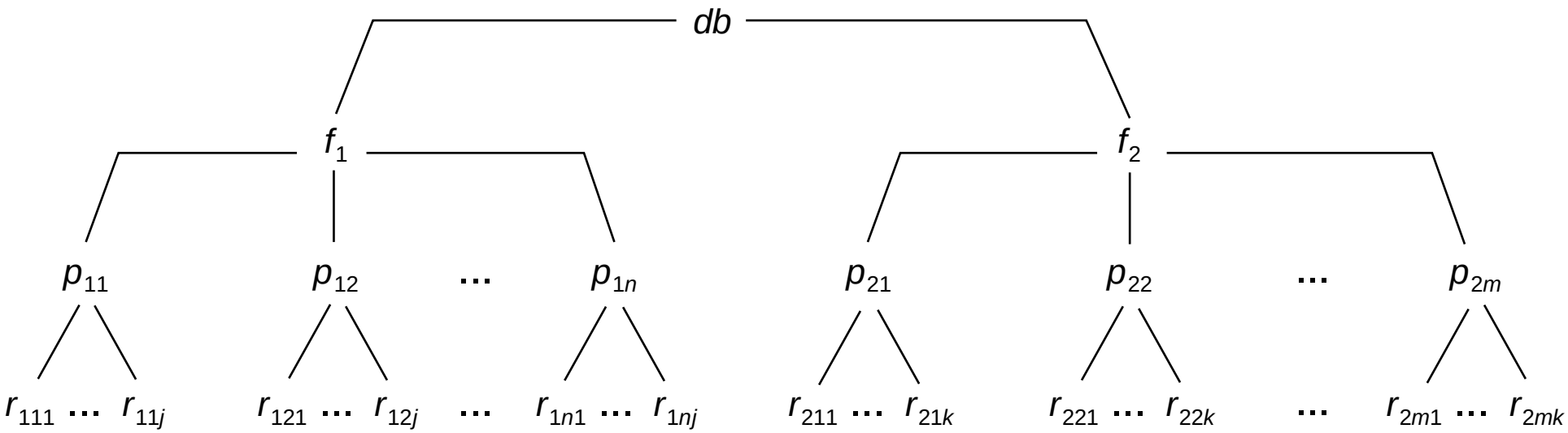

Figure 20.8 Lock compatibility matrix for multiple granularity locking.

	IS	IX	S	SIX	X
IS	yes	yes	yes	yes	no
IX	yes	yes	no	no	no
S	yes	no	yes	no	no
SIX	yes	no	no	no	no
X	no	no	no	no	no

T_1	T_2	T_3
IX(*db*)		
IX(f_1)		
	IX(*db*)	
		IS(*db*)
		IS(f_1)
		IS(p_{11})
IX(p_{11})		
X(r_{111})		
	IX(f_1)	
	X(p_{12})	
		S(r_{11j})
IX(f_1)		
IX(p_{21})		
X(r_{211})		
unlock(r_{211})		
unlock(p_{21})		
unlock(f_2)		
		S(f_2)
	unlock(p_{12})	
	unlock(f_1)	
	unlock(*db*)	
unlock(r_{111})		
unlock(p_{11})		
unlock(f_1)		
unlock(*db*)		
		unlock(r_{11j})
		unlock(p_{11})
		unlock(f_1)
		unlock(f_2)
		unlock(*db*)

Figure 20.9 Lock operations to illustrate a serializable schedule.