

# Course 1-02-326: Database Systems Engineering

## Project 1: Transaction Conflict Detection

Instructor: Michael J. May

Due 7 March 2011

### 1 Introduction

We have discussed in class several kinds of conflicts which can arise between transactions. The most common transaction types are: Read-Write, Write-Read, and Write-Write. We have also discussed the notions of View and Conflict Serializability, a property of a schedule which relates to whether its result is identical to some serial execution of the interleaved transactions. Strict Two Phase Locking (Strict 2PL) is an algorithm that we have discussed in class as well which guarantees that all interleaved schedules are serializable. Strict 2PL is quite conservative and it disallows many schedules which are in fact serializable.

In this exercise you will develop a parser which will determine automatically whether a given schedule is Serializable and whether it is permitted by Strict 2PL.

### 2 Input Description

The input for the parser will be a single file which contains an interleaved schedule of one or more transactions. To make file parsing easier, the input will be structured as follows. Consider the schedule shown for  $T_1, T_2, T_3$ :

| $T_1$  | $T_2$  | $T_3$  |
|--------|--------|--------|
| $R(A)$ |        |        |
|        | $R(B)$ |        |
| $W(C)$ | $R(C)$ |        |
|        |        | $R(C)$ |
| $R(A)$ | $W(A)$ |        |
| Commit |        | $R(A)$ |
|        | Commit |        |
|        |        | Commit |

The input file for the above schedule would be:

```
3
T1;T2;T3
3
A;B;C

11
T1:R(A)
```

```
T2:R(B)
T1:W(C)
T2:R(C)
T3:R(C)
T1:R(A)
T2:W(A)
T1:Commit
T3:R(A)
T2:Commit
T3:Commit
```

The first line is the number of transactions (3). The second line is a semicolon delimited list of the transaction names. The third line is the number of objects (3). The fourth line is a semicolon delimited list of the object names. The fifth line is blank. The sixth line is the number of entries in the events list (11). The next 11 lines are the events in the schedule in the order in which they occur. First the name of the transaction is given, then there is a semicolon, then the action (Read) or (Write) is given for the object.

Each file will contain only one schedule.

Your program should be able to handle multiple input files, one after the other. A sample input screen is shown in Figure 1.

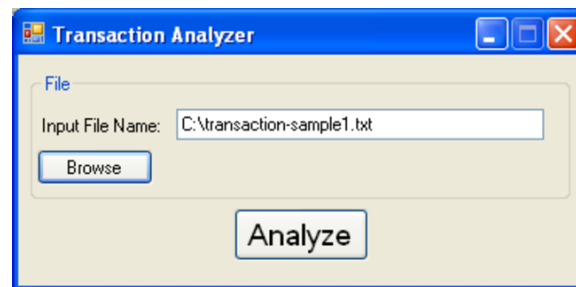


Figure 1: Input screen shot

### 3 Output

The output for the parser should be a message window indicating whether the schedule is *conflict serializable* and whether it is permitted by Strict 2PL. If the schedule is not conflict serializable, the conflicts should be listed. If the schedule is conflict serializable, print the serial schedule it is conflict equivalent to. If the schedule is not permitted by Strict 2PL, the conflicts should be listed.

For example, in the above schedule, the output window in Figure 2 should be shown:

**Hint:** One way to determine conflict serializability is by building precedence graphs for the transactions, checking if they have loops (if not then it is conflict serializable), and then performing a topological sort (to get the equivalent serial schedule).

### 4 Robustness

Your application should be able to handle the following cases without crashing:

- A badly formatted input file

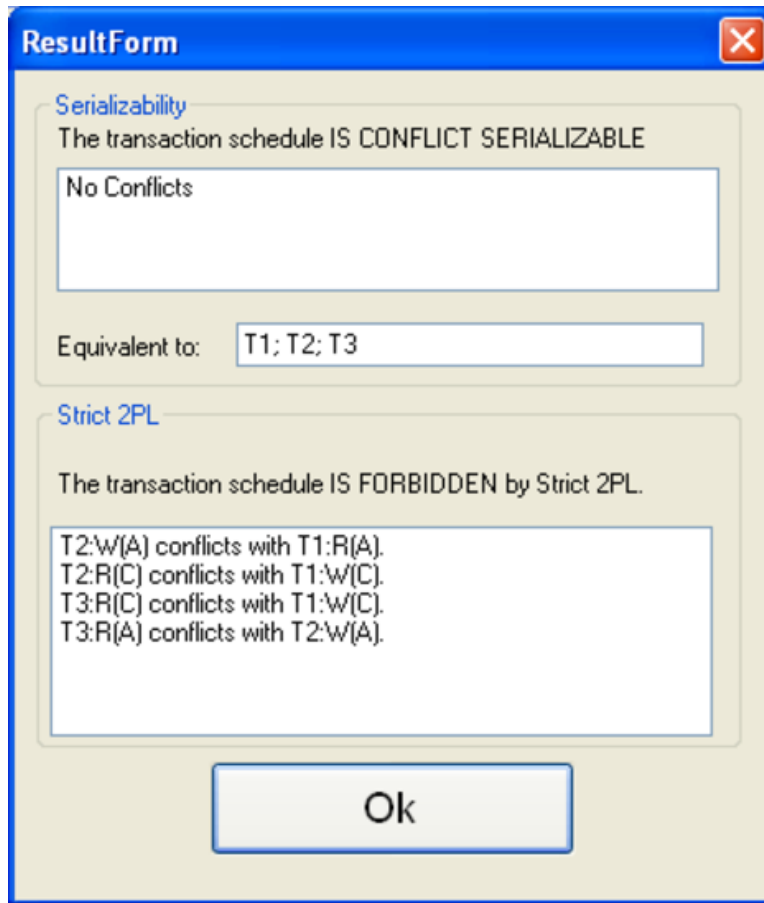


Figure 2: Output screen shot

- An input file with incorrect parameters or data
- An input file with illogical ordering of commands

In all cases, the the program should show an error window and request the user to try again. Do not try to have the program guess about the file's intent or fix the file.

#### 4.1 Extra Credit Option

**Do this only once you have completed the first part of the project**

For each schedule, determine whether it is *view serializable* and what serial schedule it is view equivalent to.

Add an line to the window indicating whether the schedule is view serializable. Print the result along with the equivalent serial schedule (if there is one).

**Note:** The problem of determining view serializability is *NP Complete*. This does not mean that it is impossible to solve, just that there is no known efficient algorithm in polynomial time. It is not too hard to write down an exponential time algorithm for the problem.

## 5 Grading and Submission

The assignment is worth 100 points total. The extra credit portion is worth an additional 25 points.

Points will be assigned as follows:

1. Correct identification of Strict 2PL and its conflicts: 30 points
2. Correct identification of Conflict Equivalence and its conflicts: 60 points
3. Robustness against failures: 10 points
4. View Serializability (if it works): +25 points extra

You may work in groups of up to 3 students.

You must submit your work to the course email ([ise326@gmail](mailto:ise326@gmail.com)) by 7 March 2011 at 11:50pm. Keep in mind the submission rules and late penalties listed in the syllabus.

Submit your work as a single RAR file with the following contents:

- The Visual Studio C# or Visual Basic project file
- All related source code files necessary to compile the project
- A compiled executable file
- A README file which contains the names of the students who worked on the project, the number of hours spent working on it, and any notes or documentation necessary to understand and use the application.