Innovations for Testing Parallel Software

Kerstin Eder

Design Automation and Verification

University of BRISTOL

Department of COMPUTER SCIENCE

SW development in industry [Martyn Thomas]

- Requirements are stated informally, usually in natural language and diagrams
  - Requirements are incomplete and contradictory
- Designs are developed using notations that have no well-defined meaning
  - Designs cannot be formally checked to meet the requirements
- Programs are written in languages that have informal specifications or that permit ambiguous programs; they provide very limited support for automatic checking for correct program flow, correct data flow, consistent use of data items, and typically have no support for checking that the program implements the design correctly or at all.
- **The main method for gaining assurance that the program is correct is to test it.**
  - *"Testing can only show the presence of bugs, never their absence."* [Dijkstra, 1972]
- When errors are found, only the program is changed.

From sequential to parallel

- Execution of a sequential program only depends on the starting state and its inputs.
- Execution of parallel programs also depends on the interactions between them.

Shared Memory:

- Communication is based on altering the contents of shared memory locations (Java).
- Usually requires the application of some form of locking (e.g., mutexes, semaphores, or monitors) to coordinate between threads.

Message Passing:

- Communication is based on exchanging messages (occam, XC).
- The exchange of messages may be carried out asynchronously, or synchronously, i.e. the sender blocks until the message is received.

The problems with parallel SW

- **Parallelism introduces non-determinism**
  - The order of execution can't be guaranteed
  - Multiple executions of the same test may have different interleavings and therefore different results
  - This can lead to unexpected program behaviour
    - e.g. race conditions, deadlock, livelock
    - very hard to reproduce and debug
  - No useful coverage models for the interleaving space

Classical Race Conditions

```
static int num = 0;

thread1 () {
int val = num;    // step 1
num = val + 1;    // step 3
}


thread2 () {
int val = num;    // step 2
num = val + 1;    // step 4
}
```

```
static int num = 0;

thread1 () {
int val = num;    // step 1
num = val + 1;    // step 2
}


thread2 () {
int val = num;    // step 3
num = val + 1;    // step 4
}
```

So why don't we just use "num++"?

Another example

```
static void transfer(Transfer t) {
   balances[t.fundFrom] -= t.amount;
   balances[t.fundTo]   += t.amount;
}
```

- Expected Behavior:
  - Money should pass from one account to another
- Observed Behavior:
  - Sometimes the amount taken is not equal to the amount received
- Possible bug:
  - Thread switch in the middle of money transfers
  - Second thread updates “Transfer t” or “balances” in parallel
- So what are the solutions?
  - Locks (mutexes, semaphores or monitors) to coordinate communication.
  - These introduce other types of bugs!

When does deadlock occur?

Coffman, Elphick and Shoshani identified **4 necessary and sufficient conditions for deadlock** to occur:

[System Deadlocks. ACM Computing Surveys 3, 2 (June), p. 67-78, 1971.]

1. Tasks claim exclusive control of the resources they require, e.g. locks (“mutual exclusion” condition).
2. Tasks hold resources already allocated to them while waiting for additional resources (“wait for” condition).
3. Resources cannot be forcibly removed from the tasks holding them until the resources are used to completion (“no preemption” condition).
4. A circular chain of tasks exists, such that each task holds one or more resources that are being requested by the next task in the chain (“circular wait” condition).

How can deadlock be avoided?

Commonly found bugs

- An operation is assumed to be atomic but it is actually not.
- Wrong or no lock
- Protocol errors
  - Mismatch between channel ends
  - Missing send or receive
- Orphaned threads due to abnormally terminating master thread

Catching these bugs is challenging

Finding bugs in parallel SW

- Research shows that bugs due to parallel code execution represent only ~10% of the bugs
- But they are the hardest to find
  - If we run the same test twice it is not guaranteed to produce the same result (non-determinism)
    - **Heisenbug**
- A disproportionate number are found late or by the customer
  - Require large configurations to test
  - Typically appear only in specific configurations
  - These bugs are the most expensive!

- **We need to have some ways of disturbing the execution**
- **We need to know when we are done**

Interesting Observations [Shmuel Ur]

- In practice thread switches are few and far between
  - The probability that the previous bug will be found is low
  - Synchronization usually operate as a no-op
    - Removing all synchronizations usually will not impact testing results!
  - Not knowing the synchronization primitives exact definition does not impact testing but the program is incorrect
    - Exception in synchronization: do you still have the lock?
    - What is the synchronization on?

- Thread scheduling for small applications is almost deterministic in a simple environment
  - Each environment has its own interleaving
  - Customer on the first day finds bugs in well tested applications!

Disturbing the execution

Philosophy

- Modify the program so it is more likely to exhibit bugs (without introducing new bugs – no false alarms)
- Minimize impact on the testing process (under-the-hood technology)
- Reuse existing tests

Techniques to instrument concurrent events

- Concurrent events are the events whose order determines the result of the program, such as accesses to shared variables, calls to synchronization primitives
- At every concurrent event, a random-based decision is made whether to cause a context switch – noise injection
  - e.g., using a sleep statement
- ConTest by IBM is an example

Knowing when we are done

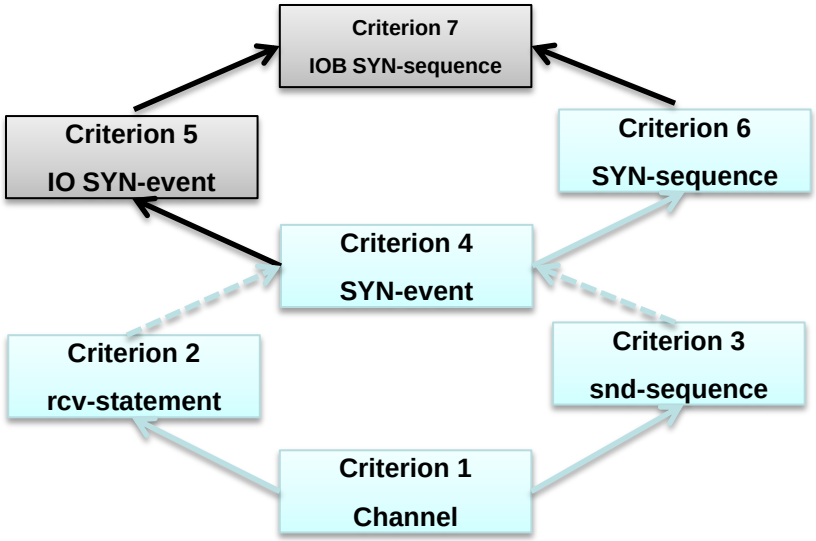
- What are our current models of test completion?
  - Black box?
    - Requirements coverage
    - Test matrix
    - Use-case coverage
    - All tests written and passing?
  - White box?
    - Code coverage at 90% or some other random number

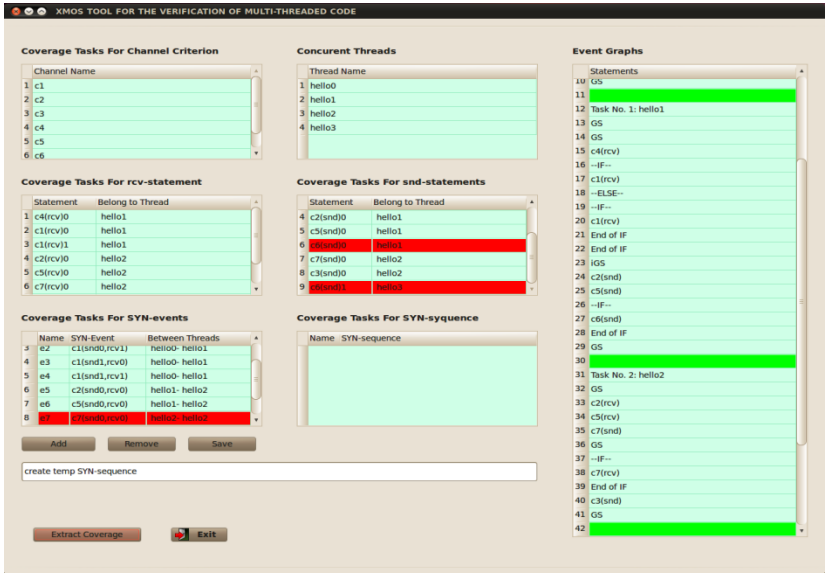
These will not be enough! Why?

New coverage models

- For shared memory:
  - Synchronization coverage
    - Make sure that every synchronization primitive was fully exercised
  - Shared variable coverage
    - Make sure shared variables were accessed by more than one thread
  - ConTest from IBM implements these coverage models
- For message passing:
  - Communication coverage for multi-threaded message passing programs

New Coverage Model [Kyriakos Georgiou, K. Eder, TVS, XMOS]





- Captures communication infrastructure in a message-passing program
  - Fully automatic extraction of coverage tasks up to SYN-events
  - User input required for SYN-sequences (tool support, semantics)
  - Complements existing code and functional coverage models
- Added value in practice
  - Bugs can be captured before even running test cases on the code
  - SYN-sequences permit testing protocol compliance

Conclusions

Testing of parallel SW has many challenges:

- Non-determinism
- Race conditions, deadlocks, livelocks
- Heisenbugs

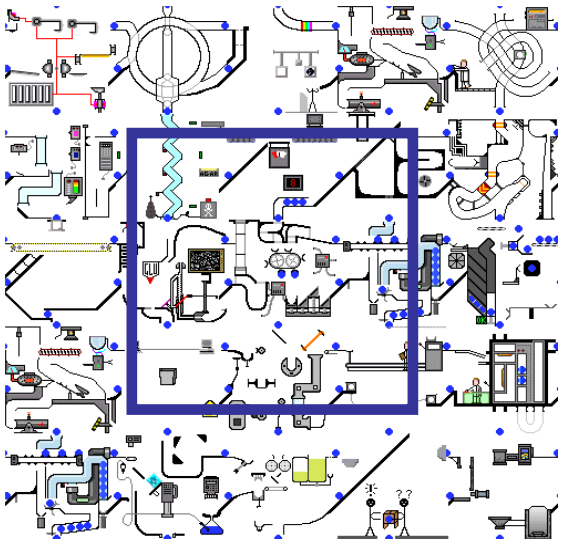
– How to provoke the 10% of bugs down to parallelism?
  - Making rare events happen more often

– How to know when you are done?
  - New coverage models

More fundamental questions

- Does testing work?
  - Is it valid to interpolate between test results?

- Does testing scale?
  - It is impossible to test all interleavings, and impossible to test any meaningful fraction of system states.
- Is testing an appropriate method to gain confidence in the correctness of complex, parallel SW?
  - What does testing tell you?
    - Useful for validation
  - Are formal methods the answer?
    - Formal verification
    - Correctness by design
      - Can we ensure no races?
      - Can we ensure no deadlocks?



Thank you

To explore new approaches towards testing parallel software e.g. in short term research or undergraduate student projects, please contact:

Kerstin.Eder@bristol.ac.uk