



Analysis of Formal Methods Used for Concurrency Control Verification

Salman Shahid
Kashif Khan

CSE 814

Formal Methods in Software Development

Sequence of Presentation

- Motivation
- Introduction
- Discussion of Covered Techniques
- Points of Application
- Identification of Orthogonality
- Possible Combinations
- Conclusion

Motivation

- To show how the challenges inherent in concurrency control verification are solved using formal methods.
- Provide a reflection of the current state of the art in the field.
- Present a qualitative analysis of the techniques discussed
- Identify & suggest ways in which these techniques can be combined to improve the chances of producing error free code.

Introduction

- To take full advantage of the parallelism characteristic of current microprocessors requires development of correct concurrent code.
- Easy to state, Difficult to achieve.
- Concurrency control involves prevention of deadlock, livelock, high level data races as well as atomicity violations amongst other things.
- Extensive body of work dealing with the problem under discussion
- Techniques discussed are from recent literature presented at noted forums and thus, can be considered to be representative of the newest direction being taken.

Methodology

- Detailed analysis of five different techniques for the verification of concurrency control.
- Categorization of each technique in specific class of formal methods.
- Identification of the specific aspects of concurrent program execution constrained or checked by each technique.
- Identification of points of commonality and orthogonality between each technique.
- Suggestion for possible combination of the techniques presented along these points.
- Enumeration of advantages of each combination.

Overview

● Static Methods

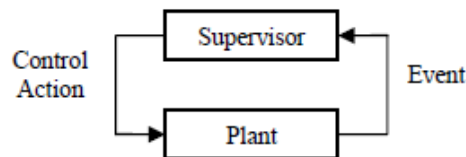
- Generation of Concurrency Control Code using Discrete-Event Systems Theory (Formal Verification)
- Static Data Race Detection for Concurrent Programs with Asynchronous Calls. (Static Analysis)

● Dynamic Methods

- Asserting and Checking Determinism for Multithreaded Programs. (Instrumentation & Verification)
- Automatic Debugging of concurrent programs through active sampling of low dimensional random projections. (Noisemaker)
- Symbolic Pruning of Concurrent Program Execution. (Dynamic Model Checking & Symbolic Verification)

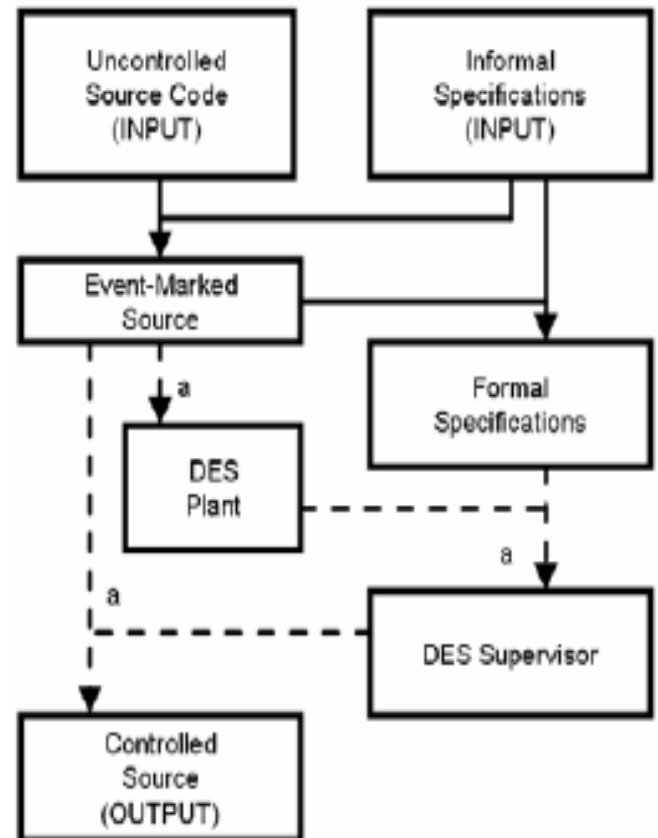
Generation of Concurrency Control Code using Discrete-Event Systems Theory (DES)

- Presented in SIGSOFT 2008.
- Goals of this technique
 - Avoiding deadlock
 - Avoiding starvation
 - Maximal subset of allowed events
- Targets automatic generation of concurrency control code by employing supervisory control synthesis.



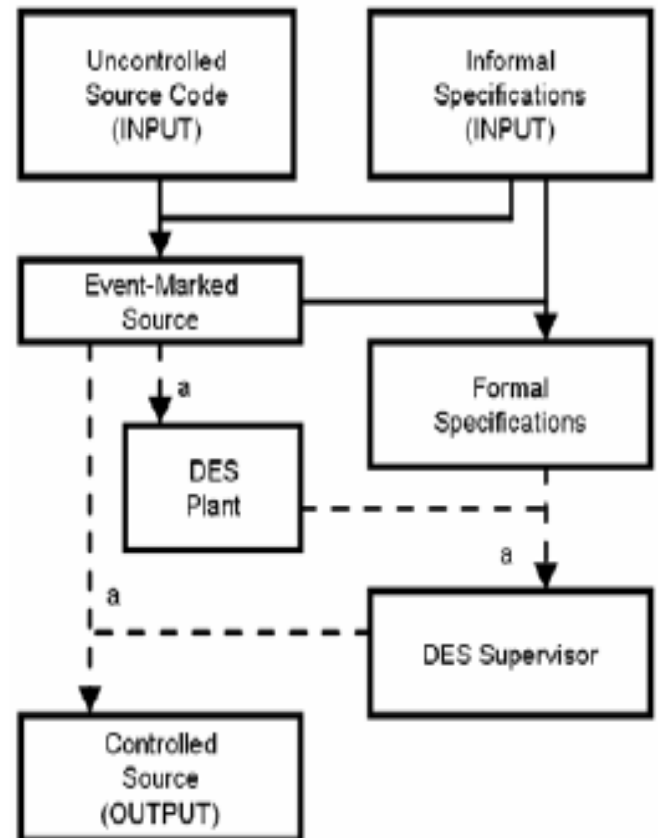
Process Steps

- Construct the set of relevant events for unsynchronized code.
- For each thread , build an FSA of all possible event sequences
 - Reduce FSA keeping only relevant events and structure preserving irrelevant events.
- Specify allowed event sequences
 - Build FSA for each restriction.
 - Combine FSA to obtain monolithic supervisor



Process Steps

- Combine supervisor and system FSA to obtain closed loop system.
- Supervisor prevents non-allowed event sequences.
- Generate code corresponding to control scheme.
 - Create semaphores in shared memory for all allowed controllable events in the initial state.
 - Controllable events enabled/ disabled by the supervisor state change function.
- Generated code weaved into original source code at points identified by supervisor.



Verification

- DES theory is proven non-blocking and correct.
- Input specifications need to be reliable
- Correctness verified through model checking. (Java Pathfinder)

Limitations

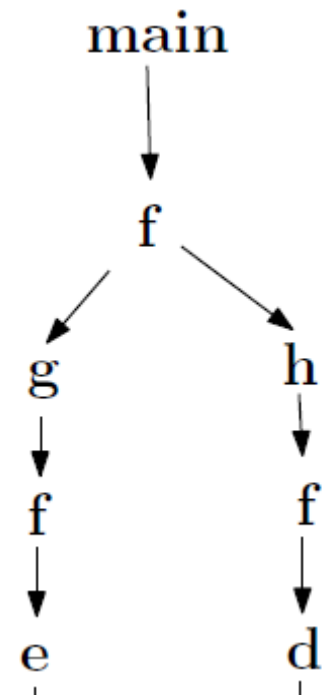
- No dynamic thread generation allowed.
- Monolithic state supervisor can suffer from state explosion.
- Formal Correctness proofs are required for guaranteed correctness.
- Starvation through livelock not explicitly addressed.

Static Data Race Detection for Concurrent Programs with Asynchronous Calls (SDRD)

- Presented in ESEC-FSE 2009.
 - Soundness
 - Preserving every real data race
 - Accuracy
 - Keeping the bogus warning low
 - Scalability
 - Being efficient in large programs
- Aims to provide static race detection concurrent programs using asynchronous calls while keeping false warning rate low.

Process

- Build Concurrent Control Flow Graph.
 - Tracks function and thread pointer for constructing CCFG.
 - Tracks lock pointers for computing locksets.
 - FSCS points to analysis is kept tractable by tracking only lock, function and thread pointers.
 - Explore only those contexts having different aliases for above pointer set.
 - Ensures reduction is data race preserving.



Process

- **Data Race Analysis**
 - Compute set of shared variable accesses for pointer variables.
 - Initial warnings based on all pairs of accesses to shared variable in different threads.
- **Pruning**
 - Compute context-sensitive locksets using CCFG.
 - Accesses can only happen by parallel threads if locksets are disjoint.
 - Thread order analysis to further prune warnings.

Discussion

- Domain specific. Only for programs using asynchronous calls.
- The algorithm uses Steensgard's algorithm for points-to analysis, which may result in possible pointers that may point to same objects (variable or function)
- The algorithm assumes that number of statements affecting locks, functions and threads are small.

Asserting and Checking Determinism for Multithreaded Programs (AD)

- Presented in FSE 2009.
- Proposes framework for constraining deterministic behavior.
- Goals of this technique
 - Precise and Direct specification of deterministic behavior.
 - Greater flexibility than traditional assertion framework.
 - Specification of parallelism correctness independent of traditional functional correctness of program.
- Deterministic specification

```
deterministic assume(sinit){  
    P // Interleaved code.  
} assert( sfinal = sfinal' = sfinal'' )
```

Process

- Structure

```
deterministic assume(Pre( $s_0, s_0'$ )) {  
    P  
} assert(Post( $s_0, s_0'$ ));
```

- P is executed multiple times with different schedule.
- If it satisfies precondition “Pre” over initial states must satisfy postcondition “Post” over final states.

Overview

- Should be semantically equivalent
- Example

```
deterministic assume( $|A - A'| < 10^{-6}$  and  $|B - B'| < 10^{-6}$ ) {
```

```
C = parallel_matrix_multiply_float(A, B);
```

```
} assert( $|C - C'| < 10^{-6}$ );
```

Predicates relate pair of states from different executions.

Bridge Predicates and Bridge Assertions.

Advantages

- Easy to use.
- Bridge predicates and assertions are flexible.
- Detects all possible harmful data races
- Distinguish harmful and benign data races.

Automatic Debugging of Concurrent Programs through Active Sampling of Low Dimensional Random Projections

- Presented in ICSE 2008
- Instrument the points in program whose relative execution order can impact the result of program.
- Inject random timing noise into all points to elicit the bug.
- Identify a set of points that indicate the source of bug.
- Pose this as an active feature selection problem

Process

- Concurrent program has relevant, irrelevant and blocking points.
- Project a lower dimensional program space using random projection.
- Projected space is sampled exhaustively.
- Irrelevant and blocking points are identified and discarded.
- Use the IGS algorithm to iteratively repeat above process until a minimal set of indicative point is found.

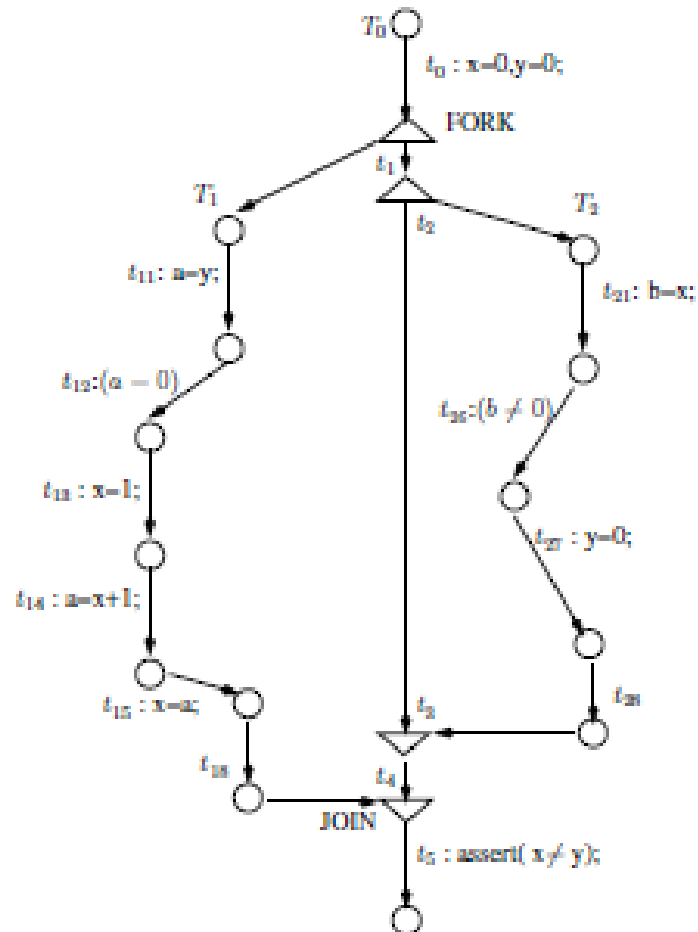
Symbolic Pruning of Concurrent Program Executions (SP)

- Presented in ESEC-FSE 2009
- Program execution traces modeled as lean partitions called Concurrent Trace Programs (CTP).
- Leverages Symbolic Methods to verify correctness in individual CTPs.
- Search space pruning using trace equivalence and property specific reduction.

Process

- Uses an enumerative algorithm to systematically generate execution traces of the program.
- Derives Concurrent Trace Programs.
 - CTP consists of partially ordered sequence of events.
 - Events in same thread ordered by their execution order in trace.
 - Events of child thread happen after fork and before join of parent thread.
 - Execution traces having same set of events have same CTPs.
 - Trace Equivalence

Example CTP



Process

- Check: Symbolically verify each CTP for property violations. Any error is guaranteed to be real error.
 - Express Verification problem as SAT formula.
 - Negate property to be checked in the formula. Thus, any assignment satisfying this would represent an error.
 - Finite size of CTP allows for representation in quantifier free first-order logic.
 - Solved by off-the-shelf Satisfiability Modulo Theory (SMT) solver.
 - If no error found then prune.

Prune

- Prune: Use a conservative analysis of the CTP to identify redundant CTPs in the future search.
 - Represent all possible traces formed from prefixes of current CTP trace including untaken branches as SAT formula.
 - Combine with SAT formula for current CTP and check for correctness.
 - Trace equivalence allows for removal of all such CTPs from future search if formula unsatisfiable.

Discussion

- Presents a method to combine model-checking with symbolic analysis.
- Significant improvement over pruning methods using partial order reduction.
- Flexibility in pruning to allow for practical constraints.
- Can be tailored to check for specific properties.

Points of Application

- **SDRD**

- Models concurrent flow and relevant locksets through CCFG.
- Identifies critical points in program.
- Combines CCFG with lockset-analysis and thread order analysis.

- **DES**

- Identifies relevant events/critical points in program.
- Builds concurrent model of system.
- Generates control code guaranteed to be free from deadlock.

Points of Application

- **Asserting Determinism.**
 - Instruments code at critical points by adding determinism check.
 - Uses instrumentation to verify determinism between parallel executions of same code.
 - Differentiates between benign and harmful data races.
 - Specification of correctness of parallelism in program.
- **Automatic Debugging**
 - Instruments points at critical points by randomly adding noise.
 - Minimizes set of points required to elicit an error.

Points of Application

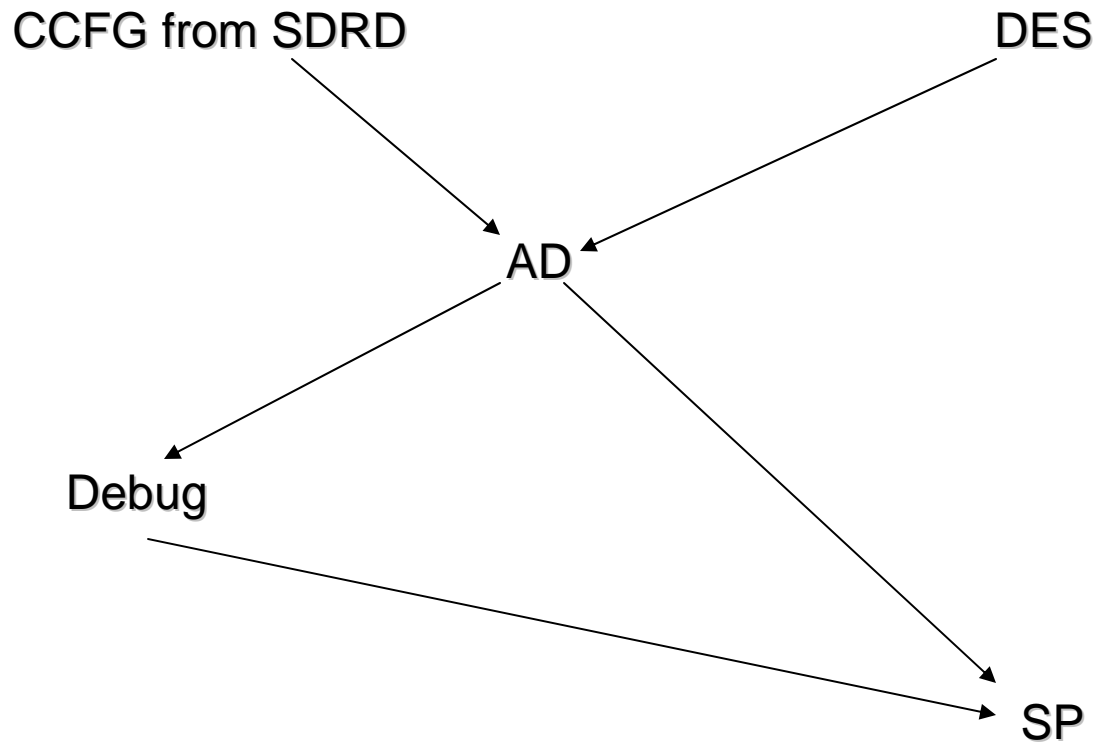
- **Symbolic Pruning.**
 - Builds a model of concrete execution trace.
 - Checks trace for correction.
 - Reduces the number of verifications required to check correctness of entire program by constraining future race checks.

Orthogonality

- Based on the points enumerated above we make the following observations:
 - SDRS cannot be generalized for all concurrent programs in its current form
 - CCFG construction can be leveraged for identification of critical points used in other schemes.
 - AD is orthogonal to all others and can be combined with each without conflict.
 - Symbolic Pruning can be used at the end of each combination to check for specific property violations.
 - Automatic debugging should be used either in parallel or prior to symbolic pruning.
 - SDRS can be the first step in using a combination of these methods only for concurrent programs with asynchronous calls.

Possible Combinations

- Tree.



Based on points identified, the following tree can be used to identify possible combinations.

Combinatorial Advantages

- Relevant Point Identification
 - DES or CCFG
- Deadlock Avoidance – (After DES).
- Identification of high-level data race – (After AD).
- Identification of Code Block with bug – (After Debug)
- Exhaustive and property specific error check – (After SP).

Conclusion

- Methods can be combined to re-enforce the advantages of each.
- Code instrumentation (AD, Debug) can lead to “observer effect”.
- Requires development of standardized interface to allow various techniques to interact practically.

References

- Generation of Concurrency Control Code using Discrete-Event Systems Theory. Dragert, Jingel et al. SIGSOFT 2008.
- Static Data Race Detection for Concurrent Programs with Asynchronous Calls. Kahlon, Sinha, Kruus, Zhang ESEC-FSE '09.
- Asserting and Checking Determinism for Multithreaded Programs. Burnim, Sen ESEC-FSE '09.
- Automatic debugging of concurrent programs through active sampling of low dimensional random projections ICASE '09.
- Symbolic Pruning of Concurrent Program Executions. ESEC-FSE'09.
- Toward a Framework and Benchmark for Testing Tools for Multi-Threaded Programs. Concurrency and Computation: Practice & Experience '05.