Software Model Checking: Locks and MLoCs

Aarti Gupta

Ph.D. from CMU SCS – Thanks Ed!

Acknowledgements: NEC Labs America
Gogul Balakrishnan, Malay Ganai, Franjo Ivančić, Vineet Kahlon, Naoto Maeda, Nadia Papakonstantinou, Sriram Sankaranarayanan, Chao Wang, Varvel group in NEC Japan
Verification Research at NEC Labs

Layered approach
- Constraint solvers
- Verification and analysis methods
- Modeling techniques

Application domains (so far)
- Software programs, multi-threaded programs
- Hardware/embedded/hybrid systems

Modeling Techniques
- Abstraction/Refinement
- Model transformations
- Modular methodology

Verification Methods
- Model checking, BMC
- Abstract Interpretation
- Dataflow analysis

Constraint Solvers
- SAT Solvers, SMT Solvers
- LP Solvers, BDDs
API Usage Bugs
- SLAM [Ball & Rajamani 01]
- Blast [Henzinger et al. 02]
- SatAbs [Clarke et al. 04]
- F-Soft-CEGAR [JIGG 05]

Does not scale for finding memory-safety bugs:
- null pointer derefs
- array buffer overflows
- string usage bugs
- uninitialized variables

If concrete model is missing alias information, CEGAR loop makes no progress.

Number of alias predicates blows up, harder to get proof.

Precise memory & pointer models:
[AGGI+ 04, ISGG+ 05]
Scalability: Finding Bugs using Search

Bounded Model Checking (BMC)
Unroll transition relation $T$ to depth $n$

Software Bounded Model Checking
Unroll program $n$ blocks

$\text{SAT solver searches space } \textit{relevant } \text{to property } p$

State sets are not saved

Critical for scalability
Finding Proofs: Scalability and Precision

Predicate abstractions: SLAM, Blast, SatAbs

Numeric abstract domains: Astrée [Cousot&Cousot 77, Blanchet+ 03]

Precision vs. Scalability

allows disjunctions

scalability challenge

path-sensitive – has precision

no disjunctions (generally)

scales well

path-insensitive – loses precision at merge

Precision: Path-Sensitive Analysis

Takes branch conditions into account

May not get proof otherwise

Q: Scalability + Path-sensitivity?
A: Lazy path sensitivity
Balancing Precision and Scalability

SAT-encoded program graph → Precise analysis with proof generalization → Path program

Learning → SAT-encoded Boolean abstraction → Theory model

Satisfiability Modulo Path Programs (SMPP) [HSIG 10]

Satisfiability Modulo Theories (SMT) [Ganzinger et al. 04, Barrett et al. 09]
F-Soft Verifier

Concrete Model → Abstract Model → Abstract Domain Analysis (F-Soft SMPP) → Bounded Model Checking (F-Soft BMC) → Concrete (true) Counterexample

CEGAR Loop:
- CEGAR loop makes no progress
- Number of predicates blows up

Refined Model → Spurious Counterexample

Precision vs. Scalability
Bugs can be deep from `main()`

**Challenges**
- Verifier runs out of time/memory
- Missing code for functions (libraries)
- Code with deep recursion (e.g. parsers)

**Strategy**
- Start from an intermediate function `foo()`

**Issue:** How to supply the environment for `foo()`?
In Practice

Environment Model

- **main()**
- **foo()**
- **bar()**
- **Error block**

Assume pre-foo

Assert pre-bar

Function Calls

Cutoff Model

From top
- Start from an intermediate function `foo()`
- Approximate environment model

From bottom
- Depth cutoff for bounding scope
- Approximate cutoff model

Modeling Strategy
- Light-weight static analysis
  - infers *likely* pre- and post-conditions, stubs

**Depth Cutoff with Design Constraints**

- Modular assume-guarantee verification
- links multiple levels in call-graph

[IBGS+ 11]

Handling MLoCs
Staging the Analyses

In-house NEC Product: Varvel
Software Factory: since Nov ’10

In 2013, Varvel applied on 65 projects, total: 40.5 MLoC, size: 1K to 20 MLoCs

Build-analyzer (works on makefiles): MLoC C/C++

- Design constraint inference: compilable units, 100s KLoC, ~10 min (1 hr timeout)
- foreach-entry-function: 10s KLoC (checked in parallel)

Depth-cutoff + model building

F-Soft SMPP
Proofs

~1 min

10 min timeout
discharges ~80% properties

F-Soft BMC
Bugs

2 min timeout
Bugs post-processed
~40% true bugs

false bugs mainly due to calling environment

Handling MLoCs
void Alloc_Page( ){
    pt_lock(&plk);
    if (pg_count >= LIMIT) {
        pt_wait(&pg_lim, &plk);
        incr(pg_count);
        pt_unlock(&plk);
        a = sh;
    } else {
        pt_lock(&count_lock);
        pt_unlock(&plk);
        page = alloc_page();
        sh = 5;
        if (page)
            incr(pg_count);
        pt_unlock(&count_lock);
        end-if
    }
}

void Dealloc_Page( ){
    pt_lock(&plk);
    if (pg_count == LIMIT) {
        sh = 2;
        decr(pg_count);
        b = sh;
        pt_notify(&pg_lim, &plk);
        pt_unlock(&plk);
    } else {
        pt_lock(&count_lock);
        pt_unlock(&plk);
        decr(pg_count);
        sh = 4;
        pt_unlock(&count_lock);
        end-if
    }
}
Data Race Detection: Staging the Analyses

CoBe: Concurrency Bench

Found ~25 critical data race bugs in 5 industry projects, 9 – 379 KLoC
Soon to be deployed in NEC’s Software Factory
Research Framework

Layered approach
- Constraint solvers
- Verification and analysis methods
- Modeling techniques

Application domains (so far)
- Software programs, multi-threaded programs
- Hardware/embedded/hybrid systems

Future domains of interest
- Distributed systems (Networks, Mobile, Cloud)
- Cyber-physical systems
- Biological systems

Beyond verification applications
- Synthesis, security, reliability
Precision–Scalability Space

Verifier Design Dimensions: (D, P, AR)

D = disjunctive state sets
!D = conjunctive state sets

P = proofs
!P = bugs only

AR = abstraction-refinement
!AR = no refinement

CBMC

F-Soft-BMC

F-Soft-CEGAR

F-Soft-SMPP

Astrée

precise modeling + model simplification

model simplification

no disjunctions
no CEGAR

lazy path sensitivity

no states
no CEGAR