Lecture 1: Assuring Software Quality by Model Checking

Edmund Clarke
School of Computer Science
Carnegie Mellon University

```c++
++CDatabase::*stats_mem_used_u
   _params.max_unrelevance = (int)
   if (_params.max_unrelevance <
   _params.max_unrelevance =
   _params.min_numClause_lits_for)
   if (_params.min_num_clause_lit
   _params.lit_num_clause_lit
   _params.max_num_clauses_le
   if (_params.lit_num_conflict_clause
   _params.lit_num_conflict_clause
   CHECK{
   cout << "Forced to reduce unre propor" << "NumRel: " " << _params
   << " MinLenDel: " " << _pd
   " MaxLenCL : " " << _pd
   }
```
Cost of Software Errors

June 2002

“Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product

... 

At the national level, over half of the costs are borne by software users and the remainder by software developers/vendors.”
“The study also found that, although all errors cannot be removed, more than a third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure that enables earlier and more effective identification and removal of software defects.”
Model Checking

- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980’s.

- Properties are written in propositional temporal logic.

- Systems are modeled by finite state machines.

- Verification procedure is an exhaustive search of the state space of the design.

- Model checking complements testing/simulation.
Advantages of Model Checking

• No proofs!!

• Fast (compared to other rigorous methods)

• Diagnostic counterexamples

• No problem with partial specifications / properties

• Logics can easily express many concurrency properties
Model of computation

Microwave Oven Example

State-transition graph describes system evolving over time.
Temporal Logic

- The oven doesn’t heat up until the door is closed.
- Not heat_up holds until door_closed
- (~ heat_up) U door_closed
Basic Temporal Operators

The symbol “p” is an atomic proposition, e.g. “heat_up” or “door_closed”.

- **Fp** - p holds sometime in the future.
- **Gp** - p holds globally in the future.
- **Xp** - p holds next time.
- **pUq** - p holds until q holds.
Model Checking Problem

Let $M$ be a model, i.e., a state-transition graph.

Let $f$ be the property in temporal logic.

Find all states $s$ such that $M$ has property $f$ at state $s$.

Efficient Algorithms: CE81, CES83
The EMC System 1982/83

- Preprocessor
- Model Checker (EMC)
- State Transition Graph: $10^4$ to $10^5$ states
- Properties
- True or Counterexamples
Model Checker Architecture

System Description

Formal Specification

State Explosion Problem!!

Model Checker

Validation or Counterexample
The State Explosion Problem

System Description

Combinatorial explosion of system states renders explicit model construction infeasible.

State Transition Graph

Exponential Growth of …
… global state space in number of concurrent components.
… memory states in memory size.

Feasibility of model checking inherently tied to handling state explosion.
Combating State Explosion

• **Binary Decision Diagrams** can be used to represent state transition systems more efficiently.  
  → **Symbolic Model Checking 1992**

• **Semantic techniques** for alleviating state explosion:
  – Partial Order Reduction.
  – Abstraction.
  – Compositional reasoning.
  – Symmetry.
  – Cone of influence reduction.
  – Semantic minimization.
<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Authors</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1981</td>
<td>Clarke / Emerson: CTL Model Checking</td>
<td>Sifakis / Quielle</td>
<td>(10^5)</td>
</tr>
<tr>
<td>1982</td>
<td>EMC: Explicit Model Checker</td>
<td>Clarke, Emerson, Sistla</td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td>Symbolic Model Checking</td>
<td>Burch, Clarke, Dill, McMillan</td>
<td>(10^{100})</td>
</tr>
<tr>
<td>1992</td>
<td>SMV: Symbolic Model Verifier</td>
<td>McMillan</td>
<td></td>
</tr>
<tr>
<td>1998</td>
<td>Bounded Model Checking using SAT</td>
<td>Biere, Clarke, Zhu</td>
<td>(10^{1000})</td>
</tr>
<tr>
<td>2000</td>
<td>Counterexample-guided Abstraction Refinement</td>
<td>Clarke, Grumberg, Jha, Lu, Veith</td>
<td></td>
</tr>
</tbody>
</table>

1990s: Formal Hardware Verification in Industry: Intel, IBM, Motorola, etc.
Model Checking since 1981

1981  Clarke / Emerson: CTL Model Checking
      Sifakis / Quielle
1982  EMC: Explicit Model Checker
      Clarke, Emerson, Sistla
1990  Symbolic Model Checking
      Burch, Clarke, Dill, McMillan
1992  SMV: Symbolic Model Verifier
      McMillan
1998  Bounded Model Checking using SAT
      Biere, Clarke, Zhu
2000  Counterexample-guided Abstraction Refinement
      Clarke, Grumberg, Jha, Lu, Veith
Grand Challenge: Model Check Software!

What makes Software Model Checking different?
What Makes Software Model Checking Different?

- Large/unbounded base types: int, float, string
- User-defined types/classes
- Pointers/aliasing + unbounded #'s of heap-allocated cells
- Procedure calls/recursion/calls through pointers/dynamic method lookup/overloading
- Concurrency + unbounded #'s of threads
What Makes Software Model Checking Different?

- Templates/generics/include files
- Interrupts/exceptions/callbacks
- Use of secondary storage: files, databases
- Absent source code for: libraries, system calls, mobile code
- Esoteric features: continuations, self-modifying code
- Size (e.g., MS Word = 1.4 MLOC)
Grand Challenge: Model Check Software!

Early attempts in the 1980s failed to scale.

2000s: renewed interest / demand:
Java Pathfinder: NASA Ames
SLAM: Microsoft
Bandera: Kansas State
BLAST: Berkeley
...
SLAM to be shipped to Windows device driver developers.

In general, these tools are unable to handle complex data structures and concurrency.
The MAGIC Tool: Counterexample-Guided Abstraction Refinement

Abstraction maps classes of similar memory states to single abstract memory states.

+ Model size drastically reduced.

- Invalid counterexamples possible.
The MAGIC Tool:
Counterexample-Guided Abstraction Refinement

C Program → Abstraction
Abstraction Guidance → Abstraction
Improved Abstraction Guidance

Abstraction → Abstract Model

Abstract Model → Verification
Yes → System OK
No → Abstraction Refinement

Abstraction Refinement → Counterexample Valid?
No → Abstraction Refinement
Yes → Counterexample Valid?

Counterexample Valid? → Abstraction Refinement
No → Abstraction Refinement
Yes → Abstraction Refinement

Abstraction Refinement → Counterexample Valid?
No → Abstraction Refinement
Yes → Abstraction Refinement

Abstraction Refinement → Abstraction
Abstraction Guidance → Abstraction
Improved Abstraction Guidance
CBMC: Embedded Systems Verification

- **Method:** Bounded Model Checking
- **Implemented** GUI to facilitate tech transfer
- **Applications:**
  - Part of train controller from GE
  - Cryptographic algorithms (DES, AES, SHS)
  - C Models of ASICs provided by nVidia
Case Study: Verification of MicroC/OS

- **Real-Time Operating System**
  - About 6000 lines of C code
  - Used in commercial *embedded systems*
    - UPS, Controllers, Cell-phones, ATMs

- **Required mutual exclusion** in the kernel
  - `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()`

- **MAGIC and CBMC:**
  - Discovered one unknown bug related to the locking discipline
  - Discovered three more bugs
  - Verified that no similar bugs are present
Case Study: Aluminum Casting Controller

• Batch metal casting control system
  – Real industrial system in use since 2001
    • 4 locations, 1000s of casts, up to 7 ingots per cast
  – 30,000 lines of C in supervisory controller

• MAGIC is able to
  – Easily verify properties of individual stages
  – Possible to check specifications spanning 5-6 stages

• Collaborating with system engineers to
  – Develop better model of process interleaving and concurrency
  – Generate additional and more detailed specifications