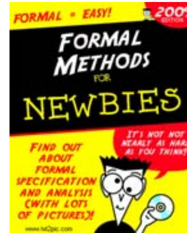
CISC422/853: Formal Methods in Software Engineering: Computer-Aided Verification



Topic 10: Software Model Checking Tool Overview

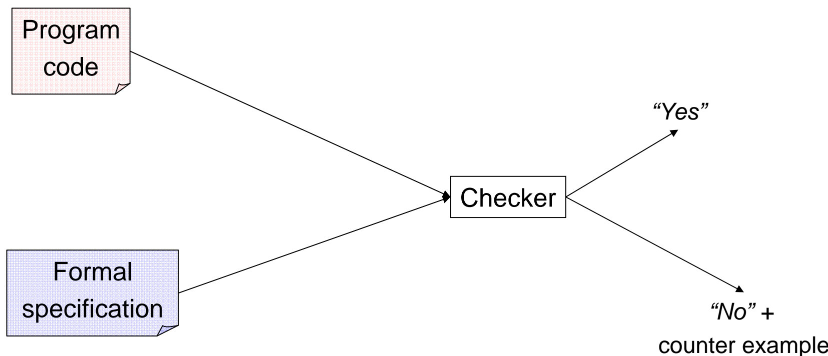
Juergen Dingel
March, 2009

Where Are We?

- How to model systems
  - **Theoretically**: FSAs
  - **Practically**: BIR, PROMELA
- How to express properties
  - Assertions, invariants
  - **Theoretically**:
    - FSA, Buechi Automata, temporal logic, LTL
  - **Practically**:
    - BIR, Never Claims, LTL
- How to check properties of systems
  - Basic DFS, BFS, nested DFS
  - Optimizations: slicing, compression, bit-state hashing, POR
- Some practical experience
  - Intuition about strengths and weaknesses of MC

Where Do We Want to Be?

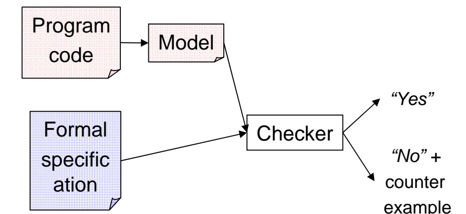
Software model checking: The Dream



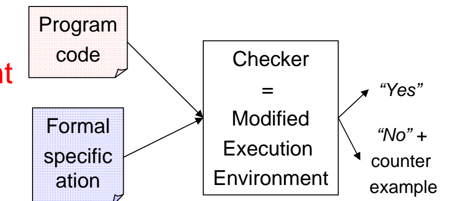
How Could We Get There?

Two classes of approaches:

- Automatic model extraction
  - Bandera/Bogor (KSU)
  - ModEx/Spin (JPL)
  - Zing (MSR)
  - Automatic abstraction refinement
    - SLAM and SDV (MSR)
    - Blast (Berkeley and EPFL)
    - Magic (CMU)

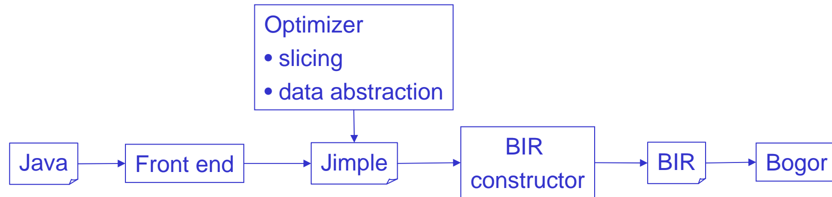


- Modified execution environment
  - VeriSoft (Bell Labs)
  - JPF (NASA Ames)
  - Chess (MSR)



Bandera/Bogor

- SW MC framework for Java developed at KSU
- Since Bandera 1.0 (alpha):
  - All of Java
  - Use Bogor (instead of Spin, SMV, …)



- Current research:
  - How to deal with native code, libraries, distributed code?
  - Distributed model checking

Bandera/Bogor (Cont'd)

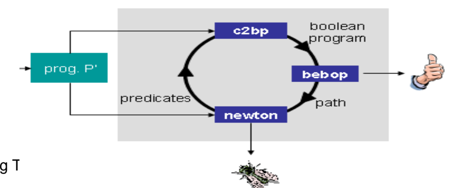
- `bandera.projects.cis.ksu.edu`
  - Code, papers, FAQ, Forum

ModEx (FeaVer) / Spin

- SW MC for distributed systems written in C
- Developed by G. Holzmann at Bell Labs (now JPL) since 1998
- Use user-defined look-up tables to translate C into PROMELA
- `cm.bell-labs.com/cm/cs/what/modex/`
  - Code
  - User guide
  - Examples
  - Papers

Automatic Abstraction Refinement (AAR)

- Problem: How to find appropriate abstraction?
- Answer: Use counter example to iteratively compute abstraction:
  - (0) start with most aggressive overabstraction P_0 of P
  - (1) if P_i satisfies property, then done
  - (2) if P_i doesn't satisfy property (w/ counter example cex), then
    - check if cex feasible in P (i.e., if cex is not a "false negative")
    - if yes, then done (P does not satisfy property, output cex)
    - if no, then
      - use cex to refine P_i into program P_{i+1} that cannot exhibit cex
      - set i to i+1 and goto 1.



Automatic Abstraction Refinement (Cont'd)

- Used by
  - SLAM/SDV (MSR)
  - Blast (Berkeley and EPFL)
  - Magic (CMU)
- Pros:
  - Appropriate abstraction computed automatically
- Cons:
  - So far, only been applied to sequential programs

SLAM, Blast and Magic

- Analyze C programs
- Predicate abstraction for abstraction refinement
- SLAM/SDV (MSR)
  - `research.microsoft.com/slam`
    - Papers
  - `http://www.microsoft.com/whdc/devtools/tools/SDV.mspx`
- Blast (Berkeley and EPFL)
  - `www-cad.eecs.berkeley.edu/~rupak/blast/`
    - Code (in Eclipse), user manual, papers
- Magic (CMU)
  - `www-2.cs.cmu.edu/~chaki/magic`
    - Code, user manual, papers

Problems With SW MC Through Translation

1. Need translation in both directions

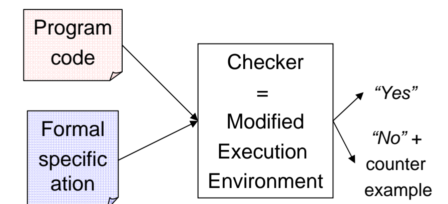
   code ⇒ model

   ⇓

   counter example (in code terms) ⇐ counter example (in model terms)
2. Correctness of analysis hinges on correctness of translation
3. Some MC languages (e.g., SMV, Spin) not well suited to represent modern, OO code
   - In Bandera, Java was initially translated into PROMELA
   - Bogor was developed to solve this problem

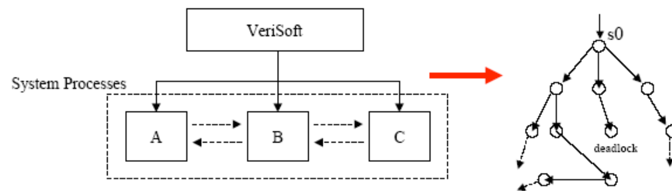
SW MC Through Modified Execution Environment

- What if run-time environment of your language knew about
  - non-determinism
  - exhaustive exploration
  - formal specifications
  - optimizations?
- You'd get
  - VeriSoft (C/C++)
  - JPF2 (Java)
  - Chess (MSR)



VeriSoft

- SW MC for concurrent C/C++ programs
- Developed by Patrice Godefroid at Bell Labs in 1996
- Analysis:
  - Directly on (only slightly modified) source code
    ⇒ no translation necessary
  - Uses VeriSoft scheduler which replaces standard C scheduler



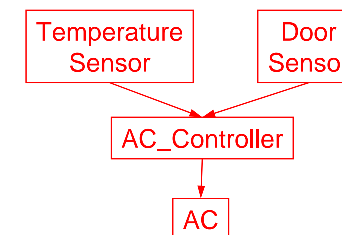
VeriSoft (Cont'd)

- Processes communicate through communication objects
  - Semaphores, channels, or shared memory
- Visible action:
  - Read or write access to communication object
- VeriSoft exhaustively enumerates all possible sequences of visible actions a concurrent program can perform up to a user-defined depth
- Supports:
  - Checks for
    - Deadlocks, livelocks, divergences, and assertion violations
  - Support for non-deterministic choice: `VS_toss(n)`
    - Simplifies implementation of test harnesses

VeriSoft (Cont'd)

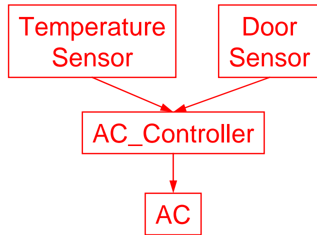
- Analysis uses
  - State-less DFS: no seen set — Wow!
    ⇒ less memory, but looping possible
    ⇒ DFS bounded by user-defined depth parameter
- Optimizations:
  - Partial order reduction
  - Search space pruning: `abort(b)` — `assume(b)` in Bogor
  - Abstraction through: placement of visible actions
- GUI allows:
  - display of computation tree up to depth
  - inspection of variable values at every node in tree
  - display of violating states
  - guided execution

VeriSoft: AC Example



"Is the AC always on, when
- *door is closed and*
- *room is hot?"*

VeriSoft: AC Example (Cont'd)



"Is the AC always on, when
- *door is closed and*
- *room is hot?"*

```
void AC_Controller() {
  char *message;
  int is_room_hot=0;        // initially, room is not hot
  int is_door_closed=1;     // and door is closed
  int ac=0;                 // so, ac is off
  while (1) {
    message=(char *)rcv_from_queue(to_me,QSZ);
    if (strcmp(message,"room_is_hot") == 0)
      is_room_hot=1;
    if (strcmp(message,"room_is_cool") == 0)
      is_room_hot=0;
    if (strcmp(message,"open_door") == 0) {
      is_door_closed=0;  ac=0;  }          // turn ac off
    if ((strcmp(message,"close_door") == 0)) {
      is_door_closed=1;
      if (is_room_hot)  ac=1;              // turn ac on
    };
  };
}
```


VeriSoft: AC Example (Cont'd)

VS_assert(b):
checks whether b is true at particular location along all possible execution paths

But how to model the environment (i.e., the sensors)?

```
void AC_Controller() {
  char *message;
  int is_room_hot=0;        // initially, room is not hot
  int is_door_closed=1;     // and door is closed
  int ac=0;                 // so, ac is off
  while (1) {
    message=(char *)rcv_from_queue(to_me,QSZ);
    if (strcmp(message,"room_is_hot") == 0)
      is_room_hot=1;
    if (strcmp(message,"room_is_cool") == 0)
      is_room_hot=0;
    if (strcmp(message,"open_door") == 0) {
      is_door_closed=0;  ac=0;  }          // turn ac off
    if ((strcmp(message,"close_door") == 0)) {
      is_door_closed=1;
      if (is_room_hot)  ac=1;              // turn ac on
    };
    if (is_room_hot && is_door_closed)
      VS_assert(ac);
  };
}
```



```
void Environment() {
  char *msg;
  msg=(char *)malloc(100);
  while (1) {
    switch(VS_toss(3)) {
    case 0: sprintf(msg, "room_is_cool");
      break;
    case 1: sprintf(msg, "room_is_hot");
      break;
    case 2: sprintf(msg, "open_door");
      break;
    case 3: sprintf(msg, "close_door");
      break;
    };
    send_to_queue(from_me, QSZ, msg);
  };
}
```

VS_toss(n):
returns value between 0 and n non-deterministically

```
void AC_Controller() {
  char *msg;
  int is_room_hot=0;        // initially, room is not hot
  int is_door_closed=1;     // and door is closed
  int ac=0;                 // so, ac is off
  while (1) {
    msg=(char *)rcv_from_queue(to_me,QSZ);
    if (strcmp(msg, "room_is_hot") == 0)
      is_room_hot=1;
    if (strcmp(msg, "room_is_cool") == 0)
      is_room_hot=0;
    if (strcmp(msg, "open_door") == 0) {
      is_door_closed=0;  ac=0;  }          // turn ac off
    if ((strcmp(msg, "close_door") == 0)) {
      is_door_closed=1;
      if (is_room_hot)  ac=1;              // turn ac on
    };
    if (is_room_hot && is_door_closed)
      VS_assert(ac);
  };
}
```


VeriSoft Simulator - Trace View
File Reset Next Event Move Go To End Quit
Process_1 Process_2
VS_toss(3)=3
send_to_queue(1,10,close_door)
rcv_from_queue(1,10)=close_door
VS_toss(3)=3
send_to_queue(1,10,close_door)
rcv_from_queue(1,10)=close_door
VS_toss(3)=1
send_to_queue(1,10,room_is_hot)
rcv_from_queue(1,10)=room_is_hot

VeriSoft Simulator - (Pruned) State-Space View
Home Zoom In Zoom Out Labels Quit
Move the pointer over a node to see which state it represents.
Key: Assertion Violations: 3 Deadlocks: 0 Aborts: 0 Branches: 40

VeriSoft Simulator
Assertion violation!
Dismiss

VeriSoft Simulator - Process 1
Step Next Continue Print Quit

```
      is_door_closed=1;
      if (is_room_hot)
        ac=1;
    };

    /* test */
    if (is_room_hot && is_door_closed)
      VS_assert(ac);
  };
}
void Environment()
{
```

VeriSoft Simulator - Process 2
Step Next Continue Print Quit

```
{
  char *message;
  message=(char *)malloc(100);
  while (1) {
    switch(VS_toss(3)) {
    case 0: sprintf(message,"room_is_cool");
      break;
    case 1: sprintf(message,"room_is_hot");
      break;
    case 2: sprintf(message,"open_door");
      break;
    case 3: sprintf(message,"close_door");
```

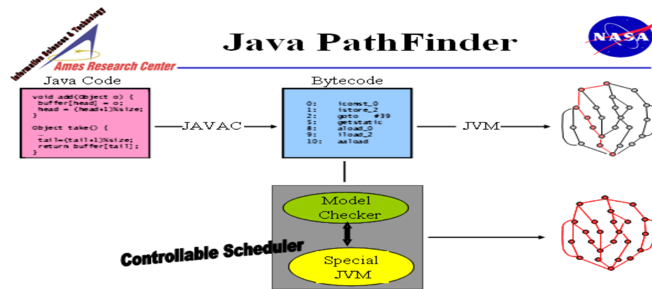
VeriSoft (Cont'd)

- Limitations:
  - No support for
    - liveness or temporal properties
    - input/output from GUIs, files, network
  - Some manual instrumentation necessary
  - No automatic abstraction techniques, e.g.,
    - data or predicate abstraction

VeriSoft Demo

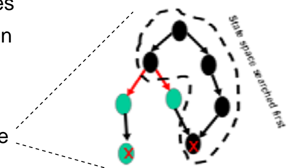
Java Path Finder (JPF)

- Model checker for Java byte code
- Developed at NASA Ames
- Analysis:
  - directly on Java byte code
    $\Rightarrow$ no translation necessary
  - uses JPF JVM



JPF (Cont'd)

- Optimizations
  - Abstraction through:
    - predicate abstraction
  - Pruning of search space:
    - **ignoreIf(b)** — **assume(b)** in Bogor
  - Heuristic search: Based on (combinations of) — inter-esting
    - property
      - deadlock: maximize # of blocked processes
      - assertion: minimize distance from assertion
    - structure of program
      - context switches increase heuristic value
      - non-determinism decreases heuristic value
    - user info
      - **interesting(b):** if b, then state receives high heuristic value
      - **boring(b):** if b, then state receives low heuristic value



JPF (Cont'd)

- Supports:
  - Checking for deadlocks, assertion violations, LTL properties
  - Non-determinism
    - `randomInt(), randomBool(), randomObject()`
  - DFS, BFS
- Limitations: No support for
  - Native code
  - Input/output through GUIs, files, network

Same as Bandera!

JPF (Cont'd)

- `http://javapathfinder.sourceforge.net`
  - Code
  - User manual
  - Papers

Course Summary

- Concurrency
  - is very tricky
  - interleavings/schedules, race conditions, deadlocks
  - testing concurrent code is difficult
- Many different ways to formally describe
  - system behaviour M
    - Theoretically: FSAs, Buechi Automata
    - Practically: BIR, Promela
  - system requirements φ
    - Safety
      - Assertions, invariants, FSAs, regular expressions, LTL, CTL
    - Liveness
      - Progress labels, Buechi Automata, Never Claims, LTL, CTL
    - Specification patterns

Course Summary (Cont'd)

- How to use state space exploration to check $M \models \varphi$
  - DFS, BFS, depth-bounded DFS/BFS, nested DFS
  - For CTL: recursive labeling algorithm (fixed point iteration)
- State space explosion
  - LTL and CTL model checking linear in size of state space, but
  - Size of state space exponential in number of processes
  - Fairness
- Optimization techniques
  - False negatives, false positives
  - Depth-bounded search, data abstraction, slicing (data flow analysis, abstract syntax trees, control flow graphs, parser generators, fixed point iteration), state and collapse compression, bitstate hashing, partial order reduction, statement merging

Course Summary (Cont'd)

- Two approaches to implementing software model checkers
  - Based on translation (Bandera/Bogor, FeaVer/Spin)
  - Based on modified execution environment (VeriSoft, JPF)
- Model checkers: Bogor, Spin

Project Presentations

1. Model checking for security (Harley)
2. Protocol conformance checking (Yann)
3. LTL-to-Buechi Automaton translation (Ruben)
4. Model checking for UML state machines (Karolina)
5. Model checking for web applications (Xianrong)

To take place on Wed, April 8 from 1:30-5pm in Goodwin 521

All are welcome

For the Final

- Monday, April 13, 2009 at 9am in Dupuis Hal 215
- 1 data sheet (8.5x11'', no flaps, no tricks)
- Take a look at CTL and CTL model checking parts of past 422 finals
- Make sure you know
  1. Concurrency: non-determinism, shared variable concurrency, the problem w/ testing
  2. Expressing behaviour: (i)FSMs, (a)synchronous composition, Buechi automata (BA)
  3. Basics of Spin & Promela, Bogor & BIR
     1. E.g., how to obtain FSM from BIR or Promela program
  4. Expressing properties: assertions, invariants, computation trees, LTL, CTL, FSMs, BAs, never claims, progress labels, safety, liveness, specification patterns

For the Final (Cont'd)

5. Analysis
   - Random, user-guided simulation
   - LTL MC: D/BFS for safety properties (possibly depth-bounded), nested DFS for liveness
   - CTL MC: labeling algorithm
   - fairness
6. Optimization: slicing, data abstraction, collapse compression, bitstate hashing, partial order reduction, statement merging

- I'm available to answer questions
- Good luck

That's all, folks!