Model Checking

Mark Greenstreet

CpSc 418 – Mar. 29, 2017

Model-Checking: Motivation

- What is “model checking”?
  - Construct a “model” for a piece of hardware or software – typically a finite-state machine.
  - Give a precise, mathematical definition of properties that the design is supposed to have.
  - Show that that model satisfies the specification.
    - For example, find all reachable states of the model.
    - Show that every reachable state satisfies a desired property – for example, mutual exclusion.
- Why use model checking?
  - Find bugs.
  - Hardware bugs are very expensive.
  - Software bugs are very common, but
    - Finding bugs in concurrent software is **hard**.
    - The challenges of finding bugs motivates using more systematic approaches.
- A simple example: Dekker’s Mutual Exclusion algorithm

Dekker's Algorithm

Problem statement: ensure that at most one thread is in its critical section at any given time.

thread 0:

```
PC0= 0: while(true) {
PC0= 1:   non-critical code
PC0= 2:   flag[0] = true;
PC0= 3:   while(flag[1]) {
PC0= 4:     if(turn != 0) {
PC0= 5:       flag[0] = false;
PC0= 6:       while(turn != 0);
PC0= 7:       flag[0] = true;
PC0= 8:     }
PC0= 9:   }
PC0=10:   critical section
PC0=11:   turn = 1;
PC0=12:   flag[0] = false;
PC0=13: }
```

thread 1:

```
PC1= 0: while(true) {
PC1= 1:   non-critical code
PC1= 2:   flag[1] = true;
PC1= 3:   while(flag[0]) {
PC1= 4:     if(turn != 1) {
PC1= 5:       flag[1] = false;
PC1= 6:       while(turn != 1);
PC1= 7:       flag[1] = true;
PC1= 8:     }
PC1= 9:   }
PC1=10:   critical section
PC1=11:   turn = 0;
PC1=12:   flag[1] = false;
PC1=13: }
```

See http://en.wikipedia.org/wiki/Dekker's_algorithm.

Is Dekker's algorithm correct?

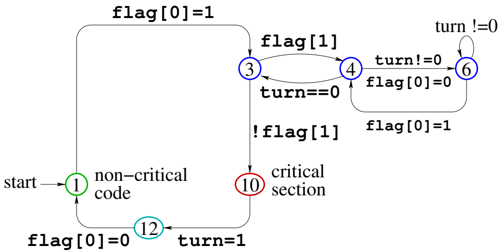
- Dijkstra (Turing Award 1972), presented the algorithm, with a proof in 1965.
- We'll use it as an example for model-checking:
  - Construct a finite-state machine model of the algorithm.
  - Determine the set of reachable states.
  - Verify that all reachable states satisfy mutual exclusion.
  - We could check other properties as well:
    - For example, freedom from starvation: show that if a process is attempting to enter it's critical region, it will eventually succeed.

Modeling Dekker's algorithm

thread 0: code

```
PC0= 0: while(true) {
PC0= 1:   non-critical code
PC0= 2:   flag[0] = true;
PC0= 3:   while(flag[1]) {
PC0= 4:     if(turn != 0) {
PC0= 5:       flag[0] = false;
PC0= 6:       while(turn != 0);
PC0= 7:       flag[0] = true;
PC0= 8:     }
PC0= 9:   }
PC0=10:   critical section
PC0=11:   turn = 1;
PC0=12:   flag[0] = false;
PC0=13: }
```

thread 0: state machine



- Each process has six control locations.
- There are three global boolean variables: `flag[0]`, `flag[1]`, and `turn`.
- This produces a total of $6^2 \cdot 2^3 = 288$ possible states.
- We want to show that no reachable state has both processes in location 10, the critical section.

Model Checking Dekker's algorithm

- Represent each state with 9-bits:
  - three for the location of each process (6 locations)
  - three for `flag` and `turn`
  - I'll show a simple python version that uses python tuples
- Pseudo-code:

```
initialState = (1,1,0,0,0); // (loc0, loc1, flag0, flag1, turn)
workList = queue(); // initially empty
knownStates = set(); // initially empty
workList.insert(initialState);
while len(workList) > 0:
   s = workList.removeNext();
   for s' in next_states(s):
      if s' not in knownStates:
         check s' for mutual exclusion;
         add s' to workList and knownStates
```

- Model-checking finds 48 reachable states for Dekker's algorithm and verifies mutual exclusion.

A Brief History of Model Checking

- Proposed by Clarke and Emerson (1981) and independently by Sifakis (1982).
  - They shared the 2007 Turing Award.
  - Their approach was essentially the one described above.
- Symbolic methods introduced by McMillan (1987) using binary-decision diagrams, a DAG representation of boolean formulas.
- Widespread adaptation of model-checking for hardware design took place in the 1990s and continues today.
  - The murφ model checker is a landmark in this work.
- Model-checking of software is now gaining industrial acceptance
  - Based on “predicate abstraction” methods of Clarke and Grumberg, and independently Ball.
  - Enabled by advances in boolean SAT solvers and interpolation-based model checking (McMillan).

Today's Paper

Protocol Verification as a Hardware Design Aid

Mark's standard five questions:

1. What problem does the paper address?
   - Hardware designs consist of large blocks that communicate using **protocols**. Mistakes in the protocol design can cause subtle errors that only occur in rare corner cases. Such errors are hard to find by traditional simulation.
2. What is the key idea in the paper?
   - Use model checking to exhaustively verify small versions of the design.
3. How do the authors validate their idea?
   - They implemented a model checker.
   - This included defining a modeling language so that protocols can be described easily and clearly.
   - They applied their approach to two protocols from real designs in industry.
4. Is the paper convincing?
   - **Yes:** they showed that they could check important properties of two "down scaled" protocols.
   - **No:** the protocols seem down-scaled to the edge of being trivial.
   - **20/20 hindsight: Definitely!** Model-checking methods have evolved and matured and are now widely used in industry for both hardware and software.
5. Any other comments?
   - Glad you asked. See the rest of the lecture.

Overview of the paper

- How does model the hardware?
  - murφ: a **guarded command language**.
- How do we state the properties to be verified?
  - Add assertions to the murφ program.
  - Use model checking to show that these assertions hold for **all** states of **all** executions.
- How do we perform the model checking?
  - Compile the murφ program to C++.
  - Link with an efficient implementation of a model checking algorithms like the one in dekker_mc.py.
  - Run the model checker to either verify the properties or report counter-examples.

murφ: a guarded command language

- In murφ a guarded command is called a rule and is written:
  `rule` *guard* `=>` *action*
  - When *guard* is satisfied, *action* **may** be performed.
  - Example: `rule ((loc[0] == 3) and flag[1]) => loc[0] := 4`
- Rules may be quantified using the `Ruleset` construction:

```
Process:  scalarset(2);
ruleset i:  Process do
   (loc[i] == 3) and flag[1-i] => loc[i] := 4
end
```

murφ: execution model

- A program defines a fixed set of rules.
- Toss all the rules in a bag.
- Repeat indefinitely:
  - Pick a rule from the bag.
  - If it's guard is satisfied, perform it's action.
  - Put the rule back in the bag.
- For verification: an "adversary" picks the rules from the bag.

Model checking today

- Lots of progress on handling larger models:
  - Symbolic methods
  - Exploit common model properties: symmetry, commuting-actions, verifiable abstraction
  - Moore's law: faster machines, larger memories.
  - Parallelism (Friday's lecture)
- Applications
  - An essential part of cache-protocol design. Used in many other aspects of hardware design as well.
  - Software: Microsoft uses model checking to verify that driver code conforms to kernel usage rules.
  - Software: Amazon uses model-checking to verify protocols used in their cloud services.
  - Many others.

Preview

March 31: The PReach Model Checker	
Reading:	Industrial Strength . . . Model Checking
April 3: Distributed Termination Detection	
April 5: Party: 50th Anniversary of Amdahl's Law	

Review

I'll add some review questions.