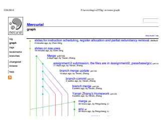
CSC 255/455

Model Checking

Instructor: Chen Ding



22 branches	22 branches
613 commits	544 commits
1251 files	1105 files
1881 changes	1679 changes
4/28/2014	4/21/2014

Model Checking

- Why static solutions?
  - what are non-static solutions called?
- Model checking
  - exhaustive checking whether an error state is reachable
- Temporal logic
  - an interface to specify properties such as "always" and "eventually"
  - how are data flow properties different?
- Symbolic model checking
  - powered Microsoft's Static Driver Verifier
    - Windows DDK

A Program as a Transition System

- Labeled transition system T = (S, I, R, L)
  - S = set of states
  - I = set of initial states
  - R in S x S = transition relation
  - L: S -> 2^AP = labeling function
- AP: set of atomic propositions
  - a proposition is variable x = y
  - the labeling function labels each state with the set of true propositions

An Example Concurrent Program

- A simple concurrent mutual exclusion program
- Two processes execute asynchronously
- There is a shared variable turn
- Two processes use the shared variable to ensure that they are not in the critical section at the same time
- Can be viewed as a "fundamental" program: any bigger concurrent one would include this one

```
10: while (true) {
11:   wait(turn == 0);
      // critical section
12:   work(); turn = 1;
13: }

|| // concurrently with

20: while (true) {
21:   wait(turn == 1);
      // critical section
22:   work(); turn = 0;
23: }
```

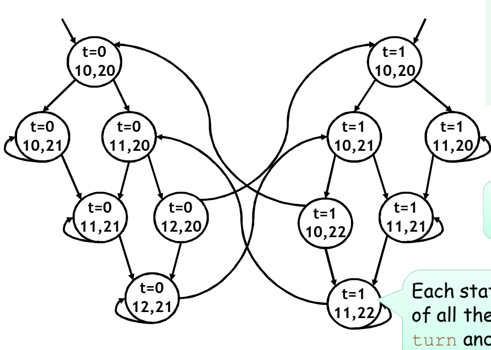
http://www.cs.colorado.edu/~bec/courses/csci5535-s10/schedule.html

Reachable States of the Example Program


```
10: while (true) {
11:   wait(turn == 0);
      // critical section
12:   work(); turn = 1;
13: }

|| // concurrently with

20: while (true) {
21:   wait(turn == 1);
      // critical section
22:   work(); turn = 0;
23: }
```

Next: formalize this intuition …

Each state is a valuation of all the variables: turn and the two program counters for two processes

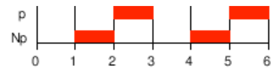
http://www.cs.colorado.edu/~bec/courses/csci5535-s10/schedule.html

Execution Paths

- A path is a sequence of states
  - each pair is a transition in R
- Linear time logic (LTL)
  - properties defined on a single path
    - e.g. path |= F p
- Computation tree logic (CTL)
  - path quantifiers
    - A: for all paths
    - E: there exists a path
  - properties defined on states
    - state |= AG p
- The expressive powers of LTL and CTL are incomparable

Temporal Properties

- Next step
  - X p
    - is true in the next state
- Invariant
  - G p
    - is true in a state if p is true in all subsequent paths
- Eventually
  - F p
    - is true somewhere on every path



Temporal Properties = Fixpoints

- States that satisfy AG(p) are all that are not in EF(not p)
- EF(not p) are states that can reach not p
- Compute EF(not p) as a fixed point
  - keep adding to EF(not p) until it converges
  - least fixed point
  - can be computed
- Use in falsification
  - i.e. finding a counter example where the property does not hold

SMT-based Model Checking

- Satisfiability Modulo Theories
  - SAT plus models for numbers and data structures
  - based on efficient SAT solvers
    - "Carla P. Gomes, Henry Kautz, Ashish Sabharwal, Bart Selman (2008). "Satisfiability Solvers". In Frank Van Harmelen, Vladimir Lifschitz, Bruce Porter. Handbook of knowledge representation. Foundations of Artificial Intelligence 3. Elsevier. pp. 89–134."
- Widely used for verification
  - bounded memory, good locality (compared to BDD)

URCS Seminars & Talks

Department Seminar Series

Wednesday, July 20, 2011
10:00 AM
CSB Room 703

Zijiang James Yang
Western Michigan University

SMT Based Symbolic Analysis for Concurrent Systems

Generating Data Race Witnesses by an SMT-based Analysis

Mahmoud Said[1], Chao Wang[2], Zijiang Yang[1], and Karem Sakallah[3]

[1] Western Michigan Univeristy
[2] NEC Laboratories America
[3] University of Michigan

https://cs.wmich.edu/~zijiang/pub/nfm11final.pdf

```
class Value {
    private int x = 1;
    public synchronized void add(Value v) {
        x = x+v.get();
    }
    public int get() {
        return x;
    }}
class Task extends Thread {
    Value v1; Value v2;
    public Task(Value v1, Value v2) {
        this.v1 = v1;
        this.v2 = v2;
    }
    public void run() {
        v1.add(v2);
    }}
class Main {
    public static void main (String[] args) {
        Value a = new Value();
        Value b = new Value();
        Thread t2 = new Thread (new Task(a, b));
        Thread t3 = new Thread (new Task(b, a));
        t2.start();
        t3.start();
    }}
```

Fig. 1. A Java program with data races.

The goal of our symbolic analysis is to search for witnesses among all sequentially consistent linearizations of $\mathcal{T}_\pi$ derived from the concrete execution π. We formulate the data race witness generation problem as a satisfiability problem. That is, we construct a quantifier-free first-order logic formula ψ_π such that the formula is satisfiable if and only if there exists a sequentially consistent linearization of $\mathcal{T}_\pi$ that leads to a state in which two data-conflict events are both enabled. The formula ψ_π is a conjunction of the following subformulas

$$\psi_\pi \ := \ \alpha_\pi \wedge \beta_\pi \wedge \gamma_\pi \wedge \rho_\pi$$

In Section 3 we present algorithms to encode the partial order (α_π), write-read consistency (β_π), and data race property (ρ_π) in first-order logic (FOL) formulas. In Section 4 we discuss the encoding of synchronization consistency (γ_π).

Symbolic Model Checking

- It represents state sets and the transition relation as Boolean logic formulas
- Fixed point computations manipulate sets of states by iteratively evaluating these formulas
- Use an efficient data structure
  - BDD

Binary Decision Diagrams (BDDs)

- Representation of boolean functions
  - a set is a function
- Disjunction and conjunction are at most quadratic
- Negation is constant
- Equivalence checking is constant or linear
- Image computation can be exponential