EECS 219C: Computer-Aided Verification

Binary Decision Diagrams & Models and Properties

Sanjit A. Seshia
EECS, UC Berkeley

Announcements

- HW 1 is up on the course webpage
- Project topics out tonight
- Next Monday class cancelled (grad admissions meeting)
  - Will be rescheduled
  - Think about your projects!
  - Send me the three topics you'd like to present (from the class webpage) or propose alternatives

More on BDDs

- Circuit width and bounds on BDD size
- Dynamically changing variable ordering
- Some BDD variants

Bounds on BDD Size: Warm-up

- Suppose the number of nodes at any level in a BDD is bounded above by B
- Then, what is an upper bound on the total number of nodes in the BDD?

Cross-section of a BDD at level i

- Suppose a BDD represents Boolean function $F(x_1, x_2, \ldots, x_n)$ with variable order $x_1 < x_2 < \ldots < x_n$
- Size of cross section of the BDD at level i is the number of distinct Boolean functions F' that *depend on* x_i given by

  $F'(x_i, x_{i+1}, \ldots, x_n) = F(v_1, v_2, \ldots, v_{i-1}, x_i, \ldots, x_n)$

  for some Boolean constants v_i's (in {0,1})

Circuit Width

- Consider a circuit representation of a Boolean function F
- Impose a linear order on the gates of the circuit
  - Primary inputs and outputs are also considered as "gates" and primary output is at the end of the ordering
  - Forward cross section at a gate g: set of wires going from output of g_1 to input of g_2 where $g_1 \leq g < g_2$
  - Similarly define reverse cross section: set of wires going from output of g_1 to input of g_2 where $g_2 \leq g < g_1$
  - Forward width (w_f): maximum forward cross section size
  - Similarly, reverse width w_r

BDD Upper Bounds from Circuit Widths

- Theorem: Let a circuit representing F with n variables have forward width w_f and reverse width w_r for some linear order L on its gates. Then, there is a BDD representing F of size bounded above by

$$n.2^{w_f \cdot 2^{w_r}}$$

BDD Ordering in Practice

- If we can derive a small upper bound using circuit width, then that's fine
  - Use the corresponding linear order on the variables
- What if we can't?
- There are many BDD variable ordering heuristics around, but the most common way to deal with variable ordering is to start with something "reasonable" and then swap variables around to improve BDD size
  - DYNAMIC VARIABLE REORDERING → SIFTING

Sifting

- Dynamic variable re-ordering, proposed by R. Rudell
- Based on a primitive "swap" operation that interchanges x_i and x_{i+1} in the variable order
  - Key point: the swap is a local operation involving only levels i and i+1
- Overall idea: pick a variable x_i and move it up and down the order using swaps until the process no longer improves the size
  - A "hill climbing" strategy

Some BDD Variants

- Free BDDs (FBDDs)
  - Relax the restriction that variables have to appear in the same order along all paths
  - How can this help?
  - Is it canonical?

Some BDD Variants

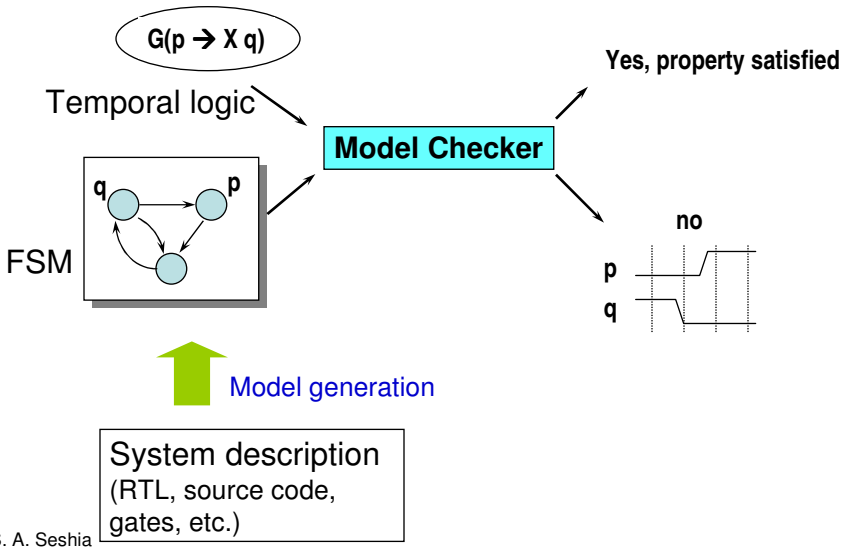
- MTBDD (Multi-Terminal BDD)
  - Represents function of Boolean variables with non-Boolean value (integer, rational)
    - E.g., input-dependent delay in a circuit, transition probabilities in a Markov chain
  - Similar reduction / construction rules to BDDs

Some BDD packages

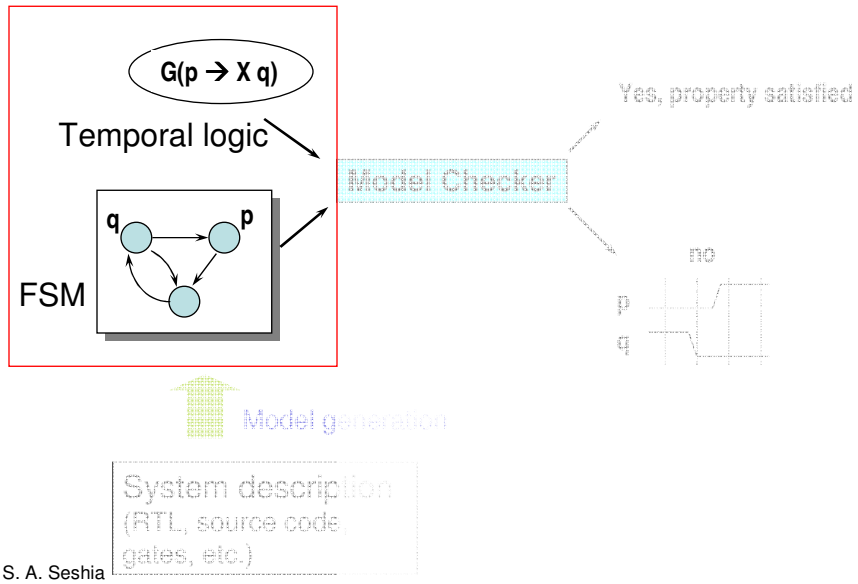
- CUDD – from Colorado University, Fabio Somenzi's group
- BuDDy – from IT Univ. of Copenhagen

Models and Properties

Finite-State Model Checking



Today's Lecture



2 Kinds of Systems

1. Open
2. Closed

- What's the difference between the two?

Verifying Closed Systems

- Assumes we have models of
  - System
  - Environment (a "good enough" one)
- Overall model is the composition of the system with its environment
- This will be the topic for most of this course

Questions addressed in this lecture

- What is a model?
- How to compose two models together?
- How to express properties of a model?

Modeling Finite-State Machines

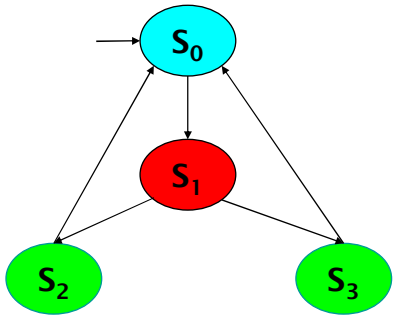
- Remember, it's a closed system – i.e., no inputs and outputs
- Common representation:
  - (S, S_0, R)
- Why do we need a *transition relation* and not just a function?
- Representation in practice:
  - (V, S_0, R)

Kripke Structure

- Alternative way of representing closed finite-state models

  (S, S_0, R, L)
  - S → set of states
  - S_0 → set of initial states
  - R → transition relation (*must be total*)
  - L → labeling function (labels a state with a set of "atomic propositions" – think of these as "colors")

Example of Kripke Structure



Why should we use Kripke structures?

Why Kripke Structures?

- Representation is independent of state-encoding
- Captures notion of "observability" to relate to actual executions
  - an observer might not be able to read all state variables

How to Compose?

- Synchronous Composition
  - All components in the system change their state variables simultaneously
- Asynchronous Composition
  - At each time point, one component changes its state
- Which form of composition exhibits more concurrency?

Specifying Properties

- Ideally, want a complete specification
  - Implementation must be equivalent to the specification
- In practice, only have partial specifications
  - Specify some "good" behaviors and some "bad" behaviors

What's a Behavior?

- Define in terms of states and transitions
- A sequence of states, starting with an initial state
  - $s_0\ s_1\ s_2\ \ldots$ such that $R(s_i, s_{i+1})$ is true
- Also called "run", or "(computation) path"
- Trace: sequence of observable parts of states
  - Sequence of state labels

Safety vs. Liveness

- Safety property
- "something bad must not happen"
- E.g.: system should not crash

- Liveness property
- "something good must happen"
- E.g.: every packet sent must be received at its destination

Examples: Safety or Liveness?

1. "No more than one processor (in a multi-processor system) should have a cache line in write mode"
2. "The grant signal must be asserted at some time after the request signal is asserted"
3. "A request signal must receive an acknowledge and the request should stay asserted until the acknowledge signal is received"

Examples: Safety or Liveness?

4. "From any state, it is possible to return to the reset state"
5. "The grant signal must be asserted 3 cycles after the request signal is asserted"

Safety vs. Liveness

- Safety property
  - Error trace is finite
- Liveness property
  - Error trace is infinite

Summary

- What we did today: BDD wrap-up, Models, Properties
- Next: Temporal logic and the simplest model checking problem