EECS 219C: Computer-Aided Verification

Properties as Automata and Explicit-State Model Checking

Sanjit A. Seshia
EECS, UC Berkeley

Mental Picture

System \[\xrightarrow{\text{trace}}\] Monitor Automaton
“checking that trace is correct”
Recap: Automata over Finite Traces

• (Regular) Finite automaton with accepting states
  – All finite traces (words) that take the automaton into the accepting state are “in its language”
• But behaviors (and traces) are infinite length
  – So we need a new notion of acceptance

Automata over Infinite Traces

• What does “Accept” mean?
  – Certain states of the automaton are called “accepting states”
  – The trace must visit an (any) accepting state infinitely often
• Such automata are called Büchi automata
  – Also Omega-automata (written ω-automata)
Example from Class

Language of the automaton = all finite-length binary strings with an odd number of 1s

Reg. expr.: $0^*1 (0 + 10^*1)^*$

If you interpret it as a Buchi automaton over infinite words: all infinite-length binary strings with an odd parity of 1s

w-regular expr: $0^*1 (0 + 10^*1)^w$

From Temporal Logic to Monitors

- A monitor for a temporal logic formula
  - is a finite automaton
  - Accepts exactly those behaviors that satisfy the temporal logic formula
    - “Accepts” means that an accepting state is visited infinitely often
- Properties are often specified as automata
Mental Picture

Automata monitoring Kripke Structures

- Recall: Trace is a sequence of the observable parts of states (labels)
- Each label is a set of atomic propositions, but can be thought of as a symbol in an alphabet
  - Alphabet is $2^{AP}$, where $AP$ is set of atomic propositions
Summary

• A (Buchi) automaton corresponding to a temporal logic formula $\phi$ accepts exactly those traces that satisfy $\phi$

Automaton for $G p$, $p$ a Boolean formula

[Diagram of a state machine labeled Start with transitions to Error labeled $p$ and $\neg p$]
Automaton for F p

Start

! p

p

Seen p

Automaton for GFp

Start

! p

! p

p

p

Seen p
From LTL to Automata

• Any LTL formula can be translated to a corresponding automaton
• There are many translation algorithms
  – We will do this later (if time permits)
• How about the other way around?
  – Can an arbitrary Buchi automaton be translated into an LTL formula?

Automaton without LTL counterpart

Automata are more expressive than LTL

What traces does the automaton below accept?

Claim: This cannot be expressed in LTL.

(How about $a \land G (a \Rightarrow X X a)$?)
On to Model Checking …

Finite-State Model Checking

Temporal logic

G(p → X q)

Yes, property satisfied

Model Checker

Explicit-State

Model generation

System description
(RTL, source code, gates, etc.)
Explicit-State Model Checking

- Model checking exhaustively enumerates the states of the system
- State space can be viewed as a graph
- Explicit-state model checking
  - Explicitly enumerates each state and traverses each edge of the graph
- We will focus on explicit-state techniques as used in SPIN [G. Holzmann, won ACM Software Systems Award]

Issues with Explicit-State MC

- The graph is usually HUGE (> $10^6$ nodes)
  - So can’t compute it a-priori
- But we are given an initial state ($s_0$) and a way of going from state to state (transition relation R)
  - In particular, we’ll assume that R is specified as a “set of actions”, each having a “enabling condition” and a “set of assignments” that cause a state change
Model Checking \( G \ p \)

- Consider the simplest property \( G \ p \)
  - \( p \) is a system invariant to be satisfied by all states
- Given the state graph, how can we check this?

- Graph traversal: DFS or BFS
Maintain 2 data structures:
1. Set of visited states
2. Stack with current path from the initial state

Potential problems?

Generating counterexamples

If the DFS algorithm finds an “error” state (in which \( p \) is not satisfied), how can we generate a counterexample trace from the initial state to that state?
Generating counterexamples

If the DFS algorithm finds an “error” state (in which \( p \) is not satisfied), how can we generate a counterexample trace from the initial state to that state?

Stack:

\[
\begin{array}{c}
\text{err} \\
\text{s1} \\
\text{err}
\end{array}
\]

Will this be the shortest counterexample?

DFS without State Set

- Only keep track of current stack
- No set of states to maintain
  - Each time you visit a state, check whether it’s on the stack
    - If so, don’t explore its edges
    - If not, do.
- Q1: Will this terminate?
- Q2: If yes: on state graph with \( n \) states, how long will it take?
Bounded Model Checking with DFS

• Same as the original DFS, except that you only allow your stack to grow up to $B$ elements deep
  – Keep track of set of all visited states and explore a state only if it is not in this set
• If this returns “no error within $B$ steps from initial state”, can you trust it?
  – NO! Example on next slide
Example

Solution: For each state, keep track of the least stack depth with which it was visited

Bound, B = 3

Breadth-First Search

- Visit states in order of distance from initial state
- Uses queue, No stack: how to generate counterexamples?
- Are the generated counterexamples the shortest?
Comparing DFS and BFS for Gp

- **Pros of BFS over DFS**
  - Shortest counterexample generated
- **Cons of BFS**
  - Need to store back-pointers to predecessor with each state in the state space representation (increased memory requirement)
  - Does not efficiently extend to liveness properties
  - Need to do cycle detection

What about non-Gp safety properties?

- **Recall:** safety properties $\rightarrow$ finite counterexample trace
- So we can construct a monitor automaton with an “error” state that must be avoided
  - Construct product of that automaton with original system
  - Error state of product has “error” in the component corresponding to the monitor