Reachability Analysis and Model Checking

Hongwei Zhang

http://www.cs.wayne.edu/~hzhang/

Ack.: this lecture is prepared in part based on slides of Lee, Sangiovanni-Vincentelli, Seshia.
The Challenge of Dependable Software in Embedded Systems

Today’s medical devices run on software… software defects can have life-threatening consequences.

[Journal of Pacing and Clinical Electrophysiology, 2004]

“the patient collapsed while walking towards the cashier after refueling his car […] A week later the patient complained to his physician about an increasing feeling of unwell-being since the fall.”

“In 1 of every 12,000 settings, the software can cause an error in the programming resulting in the possibility of producing paced rates up to 185 beats/min.”
A Robot delivery service, with obstacles

Obstacles

Starting position of robot

\( \phi = \text{destination for robot} \)

Specification:

The robot eventually reaches \( \phi \)

Suppose there are \( n \) destinations \( \phi_1, \phi_2, \ldots, \phi_n \)

The new specification could be that

The robot visits \( \phi_1, \phi_2, \ldots, \phi_n \) in that order
Reachability Analysis and Model Checking

*Reachability analysis* is the process of computing the set of reachable states for a system.

- preceding problems can be solved using reachability analysis

*Model checking* is an algorithmic method for determining whether a system satisfies a formal specification expressed in temporal logic.

Model checking typically performs reachability analysis.
Formal Verification

Property $\Phi$

System $S$

Environment $E$

Compose $M$

Verify

YES [proof]

NO counterexample
Open vs. Closed Systems

A closed system is one with no inputs

For verification, we obtain a closed system by composing the system and environment models; there are inputs/outputs between system and environment.
Model Checking $Gp$

Consider an LTL formula of the form $Gp$ where $p$ is a proposition (p is a property on a single state)

To verify $Gp$ on a system $M$, one simply needs to enumerate all the reachable states and check that they all satisfy $p$. 
Traffic Light Controller Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: count: $\{0, \ldots, 60\}$

inputs: pedestrian: pure
outputs: sigR, sigG, sigY: pure

```
M
```

```
\text{variable: count: } \{0, \ldots, 60\} \\
\text{inputs: pedestrian: pure} \\
\text{outputs: sigR, sigG, sigY: pure}
```
Model Checking $G\, p$

Consider an LTL formula of the form $Gp$ where $p$ is a proposition (p is a property on a single state).

To verify $Gp$ on a system $M$, one simply needs to enumerate all the reachable states and check that they all satisfy $p$.

The state space found is typically represented as a directed graph called a state graph.

When $M$ is a finite-state machine, this reachability analysis will terminate (in theory).

In practice, though, the number of states may be prohibitively large consuming too much run-time or memory (the state explosion problem).
Composed FSM for Traffic Light Controller

**Property:** \( G(\neg (\text{green} \land \text{crossing})) \)

This FSM has 189 states
(accounting for different values of count)

**variable:** \( \text{count: } \{0, \ldots, 60\} \)
Outline

Reachability analysis
Abstraction in model checking
Model checking liveness properties
Reachability Analysis Through Graph Traversal

Construct the state graph on the fly.

Start with initial state, and explore next states using a suitable graph-traversal strategy.
Depth-First Search (DFS)

Maintain 2 data structures:
• Set of visited states $R$
• Stack with current path from the initial state

Potential problems for a huge graph?
Generating counterexamples

If the DFS algorithm finds the target (‘error’) state $s$, how can we generate a trace from the initial state to that state?

Stack:

$S_0$

$S_1$

$S$

Simply read the trace off the stack
Explicit State Model Checking Example

Property: \( G(\neg (\text{green} \land \text{crossing})) \)

Variable: \( \text{count}: \{0, \ldots , 60\} \)

\( R = \{ (\text{red, crossing}, 0) \} \)
Explicit State Model Checking Example

**Property:**  $G(\neg(green \land crossing))$

**Variable:** count: \{0, \ldots, 60\}

\[
R = \{(\text{red, crossing, 0}), (\text{red, crossing, 1})\}
\]
Explicit State Model Checking Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: $\text{count}: \{0, \ldots, 60\}$

$R = \{ (\text{red, crossing, 0}), (\text{red, crossing, 1}), \ldots, (\text{red, crossing, 60}) \}$
Explicit State Model Checking Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: $\text{count} \colon \{0, \cdots, 60\}$

$R = \{ (\text{red}, \text{crossing}, 0), (\text{red}, \text{crossing}, 1), \cdots (\text{red}, \text{crossing}, 60), (\text{green}, \text{none}, 0) \}$
Explicit State Model Checking Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: count: \{0, \ldots , 60\}

$R = \{ (\text{red}, \text{crossing}, 0), (\text{red}, \text{crossing}, 1), \ldots (\text{red}, \text{crossing}, 60), (\text{green}, \text{none}, 0), (\text{green}, \text{none}, 1) \}$
Explicit State Model Checking Example

Property: $G(\neg(green \land crossing))$

variable: count: \{0, \ldots, 60\}

$R = \{(red, crossing, 0), (red, crossing, 1), \ldots (red, crossing, 60), (green, none, 0), (green, none, 1), \ldots, (green, none, 60)\}$
Explicit State Model Checking Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: count: $\{0, \ldots, 60\}$

$R = \{ (\text{red}, \text{crossing}, 0), (\text{red}, \text{crossing}, 1), \ldots, (\text{red}, \text{crossing}, 60),
(\text{green}, \text{none}, 0), (\text{green}, \text{none}, 1), \ldots, (\text{green}, \text{none}, 60),
(\text{yellow}, \text{waiting}, 0) \}$
Explicit State Model Checking Example

Property: \(G(\neg (\text{green} \land \text{crossing}))\)

variable: \(\text{count}: \{0, \cdots, 60\}\)

\[R = \{(\text{red, crossing}, 0), (\text{red, crossing}, 1), \ldots, (\text{red, crossing}, 60), (\text{green, none}, 0), (\text{green, none}, 1), \ldots, (\text{green, none}, 60), (\text{yellow, waiting}, 0), \ldots, (\text{yellow, waiting}, 5)\}\]
Explicit State Model Checking Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: count: \{0, \ldots , 60\}

R = \{(\text{red, crossing, 0}), (\text{red, crossing, 1}), \ldots, (\text{red, crossing, 60}), (\text{green, none, 0}), (\text{green, none, 1}), \ldots, (\text{green, none, 60}), (\text{yellow, waiting, 0}), \ldots, (\text{yellow, waiting, 5}), (\text{pending, waiting, 1}), \ldots, (\text{pending, waiting, 60})\}$
The *Symbolic* Approach

Rather than exploring new reachable states one at a time, we can explore new sets of reachable states.

- However, we only represent sets implicitly, as Boolean functions.

Set operations can be performed using Boolean algebra; Represent a finite set of states $S$ by its characteristic Boolean function $f_S$

- $f_S(x) = 1$ iff $x \in S$

Similarly, the state transition function $\delta$ yields a set $\delta(s)$ of next states from current state $s$, and so can also be represented using a characteristic Boolean function for each $s$. 
Symbolic Approach (Breadth First Search)

- Generate the state graph by repeated application of transition function ($\delta$).
- If the goal state reached, stop & report success. Else, continue until all states are seen.
The Symbolic Reachability Algorithm

**Input**: Initial state $s_0$ and transition relation $\delta$ for closed finite-state system $M$, represented symbolically

**Output**: Set $R$ of reachable states of $M$, represented symbolically

1. **Initialize**: Current set of reached states $R = \{s_0\}$

2. **Symbolic_Search()** {
   3. $R_{\text{new}} = R$
   4. **while** $R_{\text{new}} \neq \emptyset$ **do**
   5.  $R_{\text{new}} := \{s' \mid \exists s \in R \text{ s.t. } s' \in \delta(s)\} \setminus R$
   6.  $R := R \cup R_{\text{new}}$
5. **end**
8. }
Symbolic Model Checking Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: $\text{count}: \{0, \ldots, 60\}$

$\text{R}$, set of reachable states, represented by:

$(\nu_1 = \text{red} \land \nu_2 = \text{crossing} \land \text{count} = 0)$
Symbolic Model Checking Example

Property: $G(\neg(green \land crossing))$

variable: count: \{0, \ldots, 60\}

$R$, set of reachable states, represented by:

$(v_1 = red \land v_2 = crossing \land 0 \leq \text{count} \leq 1)$
Symbolic Model Checking Example

**Property:** $G(\neg (green \land crossing))$

**variable:** count: \{0, \ldots, 60\}

\[
\begin{align*}
\text{green, none:} & \quad \text{count} \geq 60 / \text{count} := 0 \\
\text{red, crossing:} & \quad \text{count} < 60 / \text{count} := \text{count} + 1 \\
\text{yellow, waiting:} & \quad \text{count} \geq 5 / \text{count} := 0 \\
\text{pending, waiting:} & \quad \text{count} \geq 60 / \text{count} := 0 \\
\text{red, crossing:} & \quad \text{count} := \text{count} + 1
\end{align*}
\]

$R$, set of reachable states, represented by:

$(v_1 = \text{red} \land v_2 = \text{crossing} \land 0 \leq \text{count} \leq 60)$
Symbolic Model Checking Example

Property: $G(\neg (green \land crossing))$

variable: count: $\{0, \ldots, 60\}$

$R$, set of reachable states, represented by:

$(v_1 = \text{red} \land v_2 = \text{crossing} \land 0 \leq \text{count} \leq 60)$

$\lor (v_1 = \text{green} \land v_2 = \text{none} \land \text{count} = 0)$
Symbolic Model Checking Example

Property: $G(\neg (\text{green} \land \text{crossing}))$

variable: count: \{0, \ldots, 60\}

$R$, set of reachable states, represented by:

$v_1 = \text{red} \land v_2 = \text{crossing} \land 0 \leq \text{count} \leq 60$

$\forall (v_1 = \text{green} \land v_2 = \text{none} \land 0 \leq \text{count} \leq 1)$

$\forall (v_1 = \text{pending} \land v_2 = \text{waiting} \land \text{count} = 1)$
Symbolic Model Checking Example

**Property:** $G(\neg (\text{green} \land \text{crossing}))$

**variable:** count: $\{0, \ldots, 60\}$

$R$, set of reachable states, represented by:

$$\forall (\nu_1 = \text{red} \land \nu_2 = \text{crossing} \land 0 \leq \text{count} \leq 60)$$

$$\forall (\nu_1 = \text{green} \land \nu_2 = \text{none} \land 0 \leq \text{count} \leq 60)$$

$$\forall (\nu_1 = \text{pending} \land \nu_2 = \text{waiting} \land 0 \leq \text{count} \leq 60)$$
Symbolic Model Checking Example

Property: \( G(\neg (\text{green} \land \text{crossing})) \)

variable: count: \{0, \cdots, 60\}

\( \text{R, set of reachable states, represented by:} \)

\( (v_1 = \text{red} \land v_2 = \text{crossing} \land 0 \leq \text{count} \leq 60) \)
\( \lor (v_1 = \text{green} \land v_2 = \text{none} \land 0 \leq \text{count} \leq 60) \)
\( \lor (v_1 = \text{pending} \land v_2 = \text{waiting} \land 0 \leq \text{count} \leq 60) \)
\( \lor (v_1 = \text{yellow} \land v_2 = \text{waiting} \land \leq \text{count} = 0) \)
Symbolic Model Checking Example

Property: \( G(\neg (green \land crossing)) \)

Variable: \( count: \{0, \cdots, 60\} \)

\[ R, \text{ set of reachable states, represented by:} \]

\[ (\nu_1 = red \land \nu_2 = crossing \land 0 \leq count \leq 60) \]
\[ \lor (\nu_1 = green \land \nu_2 = none \land 0 \leq count \leq 60) \]
\[ \lor (\nu_1 = pending \land \nu_2 = waiting \land 0 \leq count \leq 60) \]
\[ \lor (\nu_1 = yellow \land \nu_2 = waiting \land 0 \leq count \leq 5) \]
Outline

- Reachability analysis
- Abstraction in model checking
- Model checking liveness properties
Abstraction in Model Checking

- Should use simplest model of a system that provides proof of safety.
- Simpler models have smaller state spaces and easier to check.
- The challenge is to know what details can be abstracted away.
- A simple and useful approach is called localization abstraction.
- A localization abstraction hides state variables that are irrelevant to the property being verified.
Abstract Model for Traffic Light Example

Property: $G(\neg(green \land crossing))$

What’s the hidden variable?
Counterexample-guided abstraction refinement (CEGAR) by Clarke et al. 2000

- Start by *hiding* almost all state variables except those referenced by the temporal logic property.
- The resulting abstract system will have more behaviors than the original system. Therefore, if this abstract system satisfies an LTL formula $\Phi$ (i.e., each of its behaviors satisfies $\Phi$), then so does the original.
- However, if the abstract system does not satisfy $\Phi$, the model checker generates a counterexample. If this counterexample is a counterexample for the original system, the process terminates, having found a genuine counterexample. Otherwise, the CEGAR approach analyzes this counterexample to infer which hidden variables must be *made visible*, and with these additional variables, re-computes an abstraction.
- The process continues, terminating either with some abstract system being proven correct, or generating a valid counterexample for the original system.
Outline

Reachability analysis
Abstraction in model checking
Model checking liveness properties
Model Checking Liveness Properties

- A safety property (informally) states that “nothing bad ever happens” and has finite-length counterexamples.

- A liveness property, on the other hand, states “something good eventually happens”, and only has infinite-length counterexamples.

- Model checking liveness properties is more involved than simply doing a reachability analysis. See Section 15.4 for more information.
Suppose we have a Robot that must pick up multiple things, in any order

\[ \phi_i = \text{robot picks up item } i, \text{ where } 1 \leq i \leq n \]

How would you state this goal in temporal logic?
Suppose we have a Robot that must pick up multiple things, in any order

\[ \phi_i = \text{robot picks up item } i, \text{ where } 1 \leq i \leq n \]

Goal to be achieved is:

\[ F\phi_1 \land F\phi_2 \land \cdots \land F\phi_n \]
Variant: Suppose we have a Robot that must pick up multiple things, *in a specified order*

\[ \phi_i = \text{robot picks up item } i, \text{ where } 1 \leq i \leq n \]

How would you state this goal in temporal logic?
Controller Synthesis

\[ \phi_i = \text{robot picks up item } i, \text{ where } 1 \leq i \leq n \]

Goal to be achieved is:

\[ \mathbf{F}(\phi_1 \land \mathbf{F}(\phi_2 \land \cdots \land \mathbf{F}\phi_n)) \]

Consider the first part alone:

\[ \mathbf{F}(\phi_1) \]

How can we use model checking to synthesize a control strategy?
Controller Synthesis

Recall that:

\[ F(\phi_1) = \neg G(\neg \phi_1) \]

Therefore, we can construct a counterexample to:

\[ G(\neg \phi_1) \]

The counterexample is a trace that gets the robot to the desired point.
A Robot delivery service, with moving obstacles

At any time step:
Robot can move Left, Right, Up, Down, Stay Put
Environment can move one obstacle Up or Down or Stay Put
→ But only 3 times total

Can model Robot and Env as FSMs
→ Robot state: its position
→ Env state: positions of obstacles and counts
A Robot delivery service, with moving obstacles

$\phi = \text{robot delivers item to destination}$

Goal to be achieved can be stated in temporal logic

$F \phi$

How can we find a path for the robot from starting point to the destination?

→ This is an example of a “reachability problem”
Summary

Reachability analysis
Abstraction in model checking
Model checking liveness properties
Assignment

Exercise #11

- Chapter 15: Exercise 3