Automated Program Verification and Testing
15414/15614 Fall 2016
Lecture 19:
Introduction to Model Checking

Matt Fredrikson
mfredrik@cs.cmu.edu

November 9, 2016

Approaches for Demonstrating Correctness

We've seen how to prove correctness with automated assistance:

- Systematic techniques for generating **verification conditions**
- **Decision procedures** for checking these VC's
- Heuristics for inferring inductive invariants

There is still quite a bit of manual effort involved in proof development

Model checking refers to a set of techniques that:

- Take a system **model** and a **temporal specification**
- Automatically determine whether the model matches the specification

Model Checking

Developed independently by Clarke and Emerson, Queille and Sifakis in the 80's

At a high level, these techniques:

- Verify correctness by **exhaustive search** of the system's state space
- Use specifications that describe **states over time**, as they evolve according to the system's computations
- Naturally handle **concurrent systems**

Model checking has been applied to:

- Hardware and embedded systems
- Systems, device driver, and concurrent software
- Network and cryptographic protocols
- Hybrid dynamical systems

Model Checking: Strengths

There are many reasons to use MC:

- Completely automatic, so **no manual proof** burden
- Handles **partial specifications** perfectly well
- Produces **diagnostic counterexamples**, which give helpful information about problematic parts of the system
- In many cases, produces answers quickly
- For the user, reasoning about concurrent systems is no more challenging than sequential systems

Model Checking: Drawbacks

There are also some things that make it challengin:

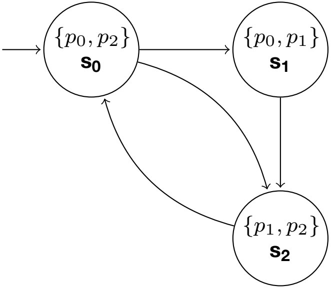
- The central issue: **state-space explosion**
- ...i.e., too many states to explore
- Another potential issue: correctness of the model

We'll look at both of these problems, and notable solutions

- State-space reduction by exploiting symmetries in the model
- Symbolic techniques that avoid exploring all states explicitly
- Program abstraction techniques that build semantically-correct models

Modeling Computation

Computations are modeled using a *state transition graph*, also called a *Kripke structure*



Kripke structure

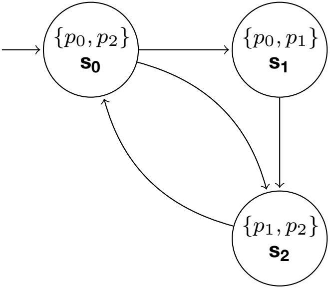
A Kripke structure $M = (P, S, I, L, R)$ consists of:

- Set of *atomic propositions* P
- States S
- Initial states $I \subseteq S$
- Labeling $L : S \mapsto 2^P$
- Transition relation $R \subseteq S \times S$

- $P = \{p_0, p_1, p_2\}$, $S = \{s_0, s_1, s_2\}$
- $I = \{s_0\}$
- $L = \{(s_0, \{p_0, p_2\}), \ldots\}$
- $R = \{(s_0, s_1), (s_1, s_2), \ldots\}$

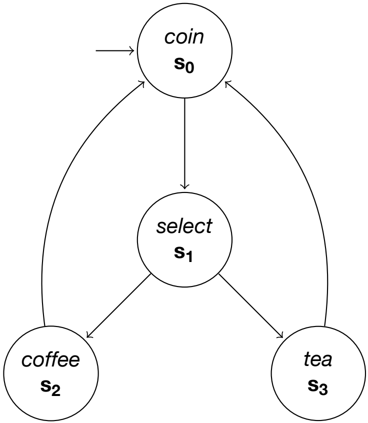
Kripke Structure: Intuition

- The atomic propositions model relevant facts about the system
- e.g., “the GPS is turned on”, $x = 5$, ...
- Transitions model the behavior of the system step-by-step
- The transition relation is **total**: for every state $s \in S$, there exists $s' \in S$ such that $(s, s') \in R$



- $P = \{p_0, p_1, p_2\}, S = \{s_0, s_1, s_2\}$
- $I = \{s_0\}$
- $L = \{(s_0, \{p_0, p_2\}), \ldots\}$
- $R = \{(s_0, s_1), (s_1, s_2), \ldots\}$

Example: Kripke Structure



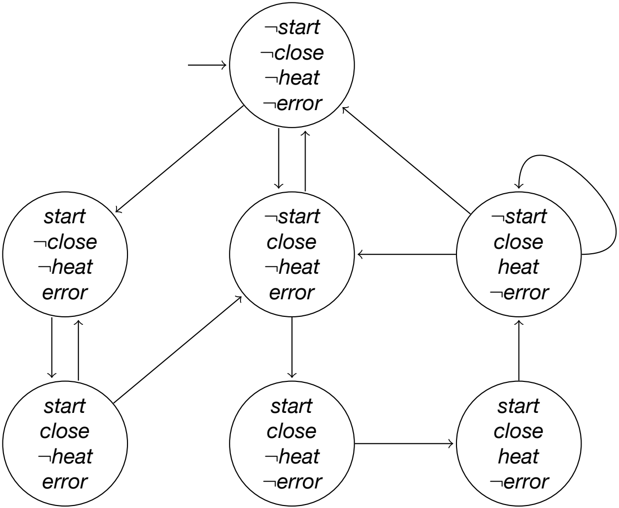
- $S = \{s_0, s_1, s_2, s_3\}$
- $P = \{\textit{coin}, \textit{select}, \textit{coffee}, \textit{tea}\}$
- $I = \{s_0\}$
- Label function:

$$L = \left\{ \begin{array}{l} (s_0, \{\textit{coin}\}), \\ (s_1, \{\textit{select}\}), \\ (s_2, \{\textit{coffee}\}), \\ (s_3, \{\textit{tea}\}) \end{array} \right\}$$

- Transition relation:

$$R = \left\{ \begin{array}{l} (s_0, s_1) \\ (s_1, s_2) \\ (s_1, s_3) \\ (s_2, s_0) \\ (s_3, s_0) \end{array} \right\}$$

Example: Microwave Transition System



Example: Deriving a Kripke Structure

Suppose we have a system with variables x, y that range over $\{0, 1\}$

It begins with $x = 1, y = 1$

The system updates its state by executing:

$$x := (x + y) \text{ mod } 2$$

Define the Kripke structure:

- $S = \{0, 1\} \times \{0, 1\}$
- $P = \{x = 0, x = 1, y = 0, y = 1\}$
- $I = \{(1, 1)\}$
- $R = \{((1, 1), (0, 1)), ((0, 1), (1, 1)), ((1, 0), (1, 0)), ((0, 0), (0, 0))\}$
- $L((a, b)) = \{x = a, y = b\}$

Modeling Computations

Computations correspond to traversals of a Kripke structure

Formally, a **path** π is an infinite sequence of states $s_0 s_1 \ldots$ where for $0 \leq i, (s_i, s_{i+1}) \in R$

Often, we'll write π^i to denote the suffix starting at i

The **trace** of π is the sequence of corresponding labels

The set of all paths forms an infinite **computation tree**

- Tree nodes represent system states
- Edges represent transitions
- Tree paths represent computations (one for each Kripke path)
- Branching results from non-determinism

Properties

We've defined the model, what are we checking?

So far, we've dealt with properties on input/output behavior

- When $P(\textit{input})$ holds, $Q(\textit{output})$ does too
- Expressed using Hoare logic, first-order assertions

In this setting, we're interested in the transition behavior **over time**

To express these, we'll use **temporal logic**

Temporal Logic: Intuition

In first-order logic, we evaluate formulas in a fixed interpretation

- The interpretation defines all facts about one particular “world”
- E.g., the predicate *polls_open* is either *true* or *false* in any world

In a temporal logic, formulas are evaluated in a **set** of worlds

- E.g., *polls_open* is true in all worlds in which the date is November 9 in an election year
- The set of worlds define **moments in time**
- Temporal operators that refer to different moments in time
- **eventually** reach a safe state; an error state is **never** reached

For us, each moment in time corresponds to a **path location**

Facts about the world come from the labeling function

Temporal Operators

There are five basic temporal operators we'll use

X p	p holds at the *next* point in time
F p	p holds at *some future* point in time
G p	p holds at *every point* in time
p **U** q	p holds *until* q holds
p **R** q	p *releases* q: q holds until p (if it ever does)

These operators describe properties of a path π

Examples: Temporal Operators

- Lunch will come eventually

$$\textbf{F}\ \textit{lunch}$$

- Requests are served until the connection terminates (if ever)

$$\textit{terminate}\ \textbf{R}\ \textit{serve}$$

- Each request is always followed by an eventual response

$$\textbf{G}\ (\textit{request} \rightarrow \textbf{F}\ \textit{response})$$

- p holds only finitely often

$$\textbf{F}\ \textbf{G}\ \neg p$$

- Whenever the start button is pressed, the oven heats eventually

$$\textbf{G}\ (\textit{start} \rightarrow \textbf{F}\ \textit{heat})$$

Linear Temporal Logic

These operators allow us to define **linear temporal logic** (LTL)

LTL contains **state formulas** and **path formulas**

State Formula

The syntax of state formulas is given by:

$$f ::= \quad \top \mid \bot \mid p \mid \neg f \mid f_1 \vee f_2 \mid f_1 \wedge f_2$$

State formulas correspond to facts that hold in a particular state.

Path Formula

An LTL formula is composed of the following elements:

$$g ::= \quad f \mid \neg g \mid g_1 \vee g_2 \mid g_1 \wedge g_2 \mid \textbf{X}\ g \mid \textbf{F}\ g \mid \textbf{G}\ g \mid g_1\ \textbf{U}\ g_2 \mid g_1\ \textbf{R}\ g_s$$

Path formulas are evaluated along a particular path.

LTL: Semantics

The semantic judgement that we use is of the form:

$$M, \pi \models g$$

Read: "g holds along path π in Kripke structure M"

We'll also use $M, s \models f$ for path formulas, where s is a state in M

The semantics of path formulas is straightforward

$$\begin{array}{lcl}
M, s \models p & \Leftrightarrow & p \in L(s) \\
M, s \models \neg f & \Leftrightarrow & M, s \not\models f \\
M, s \models f_1 \vee f_2 & \Leftrightarrow & M, s \models f_1 \text{ or } M, s \models f_2 \\
M, s \models f_1 \wedge f_2 & \Leftrightarrow & M, s \models f_1 \text{ and } M, s \models f_2
\end{array}$$

LTL: Semantics (**X** operator)

Recall, **X** p asserts that p holds in the next state

We'll replace p with an arbitrary path formula

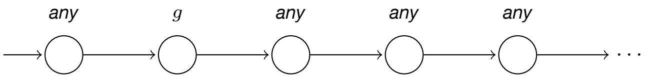
Then, we define the meaning of **X** g to be:

$$M, \pi \models \mathbf{X}\ g \quad \Leftrightarrow \quad M, \pi^1 \models g$$

(we're beginning path indices at 0)

Sometimes, it helps to visualize the path:

X g



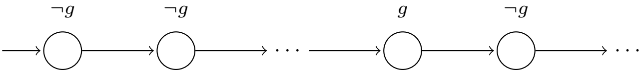
LTL: Semantics (**F** operator)

F g asserts that g holds at some point in the future

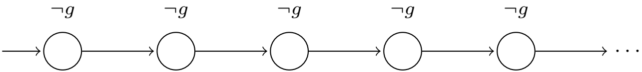
Formally:

$$M, \pi \models \mathbf{F}\ g \quad \Leftrightarrow \quad \text{exists } i \geq 0, M, \pi^i \models g$$

The following path satisfies **F** g:



Whereas this one doesn't

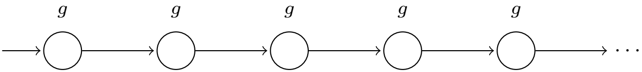


G g asserts that g holds *globally* into the future

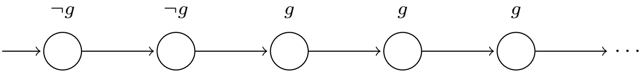
Formally:

$$M, \pi \models \mathbf{G}\ g \quad \Leftrightarrow \quad \text{for all } i \geq 0, M, \pi^i \models g$$

The following path satisfies **G** g:



Does this one?



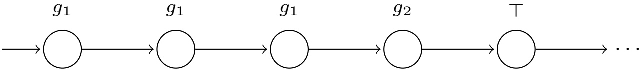
LTL: Semantics (**U** operator)

g_1 **U** g_2 asserts that g_1 holds until g_2 does

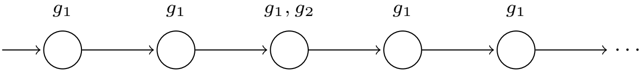
Formally:

$$M, \pi \models g_1 \textbf{ U } g_2 \quad \Leftrightarrow \quad \begin{array}{l} \text{exists } i \geq 0, M, \pi^i \models g_2, \\ \text{and for all } 0 \leq j < i, M, \pi^j \models g_1 \end{array}$$

The following path satisfies g_1 **U** g_2:



Does this one?



LTL: Semantics (**R** operator)

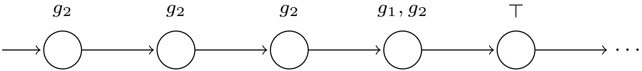
g_1 **R** g_2 asserts that g_2 *releases* g_1

It is the dual to **U**

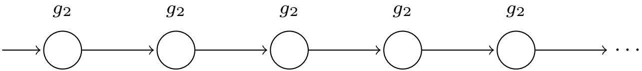
Formally:

$M, \pi \models g_1$ **R** $g_2 \quad \Leftrightarrow \quad$ forall $i \geq 0,$ if for every $j < i, M, \pi^j \not\models g_1,$ then $M, \pi^i \models g_2$

The following path satisfies g_1 **R** g_2:



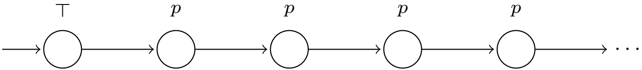
So does this one:



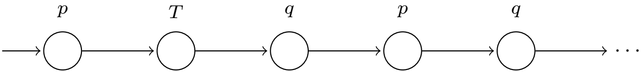
Example: Temporal Semantics

Write example paths that satisfy these formulas.

F G p



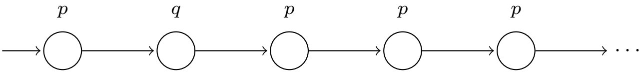
G $(p \rightarrow$ **F** $q)$



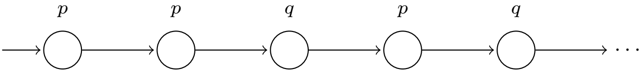
Example: Temporal Semantics

Write **counterexample** paths for satisfy these formulas.

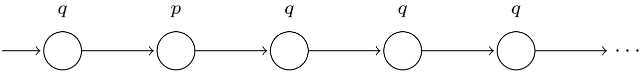
$\textbf{G}\ (p \rightarrow \textbf{F}\ q)$



$\textbf{F}\ (p \rightarrow \textbf{X}\ \textbf{X}\ q)$



$\textbf{G}\ \textbf{F}\ p$



Satisfiability and Validity of LTL

An LTL formula g is satisfiable if and only if:

there exists M where for every π in $M : M, \pi \models g$

If $M, \pi \models g$ for all π, then M is a **model** of g

An LTL formula g is valid if and only if:

for all M, π in $M : M, \pi \models g$

These notions are similar to sat. and validity we've discussed before

But notice: to be *sat*, there must be a model where **for every** π**,**

$$M, \pi \models g$$

Hence, LTL universally quantifies over all paths in the model

LTL Model Checking

LTL Model Checking

Given M and g, decide whether M is a model of g.

Formally, for $M = (P, S, I, L, R)$, decide whether for each $s_0 \in I$ and every path π starting from s_0,

$$M, \pi \models g$$

Alternatively, given $M = (P, S, I, L, R)$ find the states $s_0 \in I$ where:

$$\text{for all } \pi \text{ starting in } s_0, M, \pi \models g$$

If this set is not I, then find a path π_{cex} where:

$$M, \pi_{\text{cex}} \not\models g$$

π_{cex} is called a **counterexample**

Next Lecture

- Continue discussing model checking
- More on temporal logic
- Useful temporal properties