Verification and invariants



What are some limitations of software testing?

Safety properties and invariants

Invariant: some computable property of a system that always holds (more precise definition later)

Safety property (or safety requirement): assertion that nothing bad ever happens



How are invariants and safety properties related?

Safety properties can be expressed as invariants

Define "bad thing" computably

Invariant: bad thing is not true

Example for AC from lab 8

- 1) There is no more than 290 ms of delay between status_message messages.
- \rightarrow "bad thing": two consecutive status_message messages come more than 290 ms apart
- \rightarrow invariant: bad thing is not true
- \rightarrow your monitor checked if the invariant always held

Another example from lab 8

8) If the system is on and the control knob hasn't changed for 290 ms, the desired temperature as sent by status message obeys the formula 5400 + 25 * (control knob reading) / 8 with an error of at most 3 degrees F (300 centidegrees).

What is the "bad thing?"

Working with invariants

- Runtime monitoring on a deployed system
- Testing
- ...formally proving?

Runtime monitoring on a deployed system

Unsafe operation



Invariants for testing

Our basic understanding of testing so far has been largely **transactional**:

- Give input, observe that output matches what is expected
- Are embedded systems transactional?

Robot asked to navigate to a goal point



Image source

Formalizing invariants

...back to FSMs!

Board discussion:

Reachability

Traces

Propositional logic

Composed of terms ("a", "b", "c"), where a term can be: **p(x), q(x), r(x,y)**: propositions (evaluate to either true or false) x > 0 x + y = 2 robot x has not hit obstacle y

- **a** \bigwedge **b** : a and b (true if term a is true and term b is true)
- **a V b**: a or b (true if term a is true or term b is true or both)
- **¬a**: not a (true if term a is false)
- $\mathbf{a} \Rightarrow \mathbf{b}$: a implies b (true if term b is true or if term a is false)

Formal definition of an invariant

A property *p* of a transition system* *S* is an *invariant* of *S* if every reachable state of *S* satisfies *p*

*For our class, think of a transition system as an FSM

[Alur, chapter 3]

Inductive invariants

A property *p* of a transition system *S* is an *inductive invariant of S* if:

- 1. The initial state *s* satisfies *p*, and
- 2. If a state *s* satisfies *p*, and (*s*, *t*) is a transition, then the state *t* also satisfies *p*

(Board discussion: Prove (x >= 0 \land y > 0) V (x > 0 \land y >= 0)

Proving non-inductive invariants

To establish that a property *p* is an invariant of the transition system *S*, find a property *q* that:

- i q is an inductive invariant of S, and
- the property *q* implies the property *p* (that is, a state satisfying *q* is guaranteed to satisfy *p*)

(Board discussion: Prove B => x > O ^ y > O)

How would you deal with this invariant?

10) If the system is on and the control knob hasn't changed for 290 ms, the desired temperature as sent by status message obeys the formula 5400 + 25 * (control knob reading) / 8 with an error of at most 3 degrees F (300 centidegrees).

Stateful invariants

For a transition system *S*, Create a *safety monitor FSM* called *M* where:

- inputs of *M* are a subset of the inputs and outputs of *S*
- Some subset *E* of the states of *M* are designated as "error" states
- The behavior of *M* is designed such that if the sequence of inputs to *M* leads *M* to an error state in *E*, this is an invariant violation

Compose *M* and *S*. The invariant becomes that any state in *E* is not reachable



What similarities do you see between the safety monitor FSM definition and the runtime monitor you wrote in lab 8?