CS:4980 Foundations of Embedded Systems

Liveness Requirements

Part I

From Desktops to Cyber-Physical Systems

Traditional computers: Stand-alone devices running software applications

- e.g., data processing

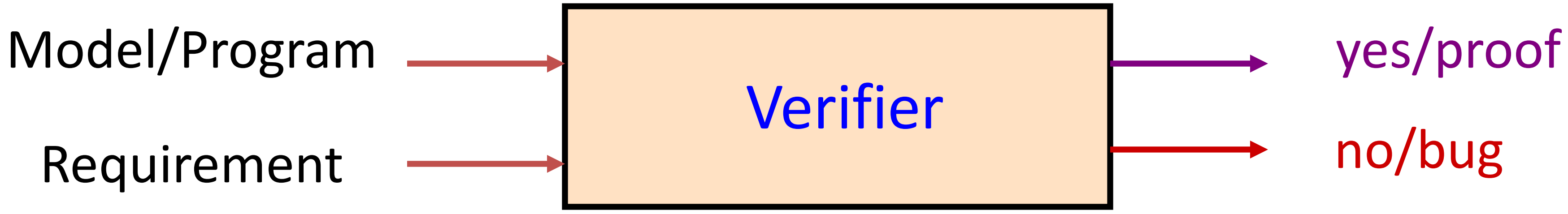
Traditional controllers: Devices interacting with physical world via sensors and actuators

- e.g., thermostat

Embedded (aka Cyber-physical) Systems: Special-purpose system with integrated microcontroller/software

- e.g., cameras, watches, washing machines, …

Formal Verification



How to formalize requirements?

1. **Safety requirements:** Invariants, monitors
2. **Liveness requirements:** Temporal logic

Recap: Safety Requirements

Nothing bad ever happens

- Trains should not be on bridge simultaneously
- If the east train is waiting, the west train should not be allowed on the bridge twice in succession

Violation of a safety property is demonstrated by a (finite) execution

Recap: Safety Requirements

Formalization:

- Identify a property φ over state variables, and check if φ is an invariant of the system
- Construct a monitor M and check that "monitor mode is not error" is an invariant of the composite system $C \parallel M$

Analysis:

- Proof based on inductive invariants
- Algorithms for exploring the reachable states of the system

Liveness Requirements

Something good eventually happens

- A waiting train is eventually allowed to enter the bridge
- Each process eventually decides to be a leader / follower

No finite execution demonstrates violation of such properties

- Counterexample should show a cycle where the system may get stuck without achieving the goal

Liveness Requirements

Formalization:

- Need to consider infinite executions (ω-executions)
- Need a logic to state properties of infinite executions

Temporal Logic

- Logics proposed to reason about time
  - Origins in philosophy
  - Tense logic: Prior (1920)
- Linear temporal logic (LTL) proposed for reasoning about executions of reactive systems
  - Pnueli (1977), later selected for Turing award (1996)
- Industrial adoption
  - Property Specification Language (PSL) IEEE standard
  - LTL enriched with many additional constructs for usability
  - Supported by CAD tools for simulation/analysis of Verilog/VHDL

Valuations and Base Formulas

V: set of typed variables

- Example: nat x, bool y

Valuation: type-consistent assignment of values to variables in V

- $q_0 : (x = 26, y = 0)$
- $q_1 : (x = 11, y = 1)$

Base formula: Boolean-valued expression over V

- even(x)
- $(y = 0) \Rightarrow \text{even}(x)$

Satisfiability: valuation q satisfies formula φ, written $q \models \varphi$, if $q(\varphi)$ evaluates to 1

- $q_0 \models \text{even}(x)$
- $q_0 \models (y = 0) \Rightarrow \text{even}(x)$
- $q_1 \not\models \text{even}(x)$
- $q_1 \models (y = 0) \Rightarrow \text{even}(x)$

Traces

A base formula expresses a property of a single valuation

Trace: Infinite sequence of valuations

- ρ : (0,0), (1,1), (2,0), (3,1), (4,0), (5,1), …
- ρ' : (0,0), (21,1), (13,1), (43,0), …

In system specification and verification:

- V can be set of state variables
  (a trace is a possible infinite *execution* of the system)
- V can be set of input and output variables
  (a trace is an observed *input/output behavior* of system)
- V can include all of state, input, and output variables

LTL Basics

A base formula expresses a property of a single valuation

Trace: Infinite sequence of valuations

LTL formulas are built from Boolean-valued expressions using

- Logical connectives: $\_ \wedge \_$, $\_ \vee \_$, $\_ \Rightarrow \_$, $\neg \_$
- Temporal operators: Always _ , Eventually _ , Next _ , _ Until _
  $\Box \_$, $\Diamond \_$, $\circ \_$, $\_ \, U \, \_$

LTL formulas are evaluated with respect to a trace

A trace $\rho = q_1, q_2, q_3, \ldots$ *satisfies* a base formula φ if $q_1 \models \varphi$

Always Operator

Always φ intuitively means φ holds at all times

For a base formula φ, a trace $\rho = q_1, q_2, q_3, \ldots$ *satisfies* Always φ if $q_j \models \varphi$ for all $j > 0$

Example: trace

```
x:  0  1  2  3  4  5  …
y:  0  1  0  1  0  1  …
```

- falsifies (i.e., does not satisfy) Always even(x)
- satisfies Always ($y = 0 \Rightarrow$ even(x))

Note: a state property φ is invariant for a transition system T iff every infinite execution of T satisfies Always φ

Eventually Operator

Eventually φ intuitively means φ holds at some point (at least once)

For a base formula φ, a trace $\rho = q_1, q_2, q_3, \ldots$ *satisfies* Eventually φ if $q_j \vDash \varphi$ for some $j > 0$

Example: trace

```
x:  0  1  2  3  4  5  …
y:  0  1  0  1  0  1  …
```

- satisfies Eventually $(y = 1)$
- satisfies Eventually $(x = 45)$
- falsifies Eventually $(x = 4 \wedge y = 1)$

Note: Eventually is the logical dual of Always: a trace

ρ satisfies Eventually φ iff ρ satisfies $\neg$Always $\neg\varphi$ iff ρ falsifies Always $\neg\varphi$

Next Operator

Next φ intuitively means φ holds the *next* time

For a base formula φ, a trace $\rho = q_1, q_2, q_3, \ldots$ *satisfies* Next φ if $q_2 \vDash \varphi$

Example: trace

```
x:  0  1  2  3  4  5  …
y:  0  1  0  1  0  1  …
```

- satisfies Next $(y = 1)$
- falsifies Next $(x = 2)$

Until Operator

φ Until ψ intuitively means ψ holds at some point and φ holds at all times until then

For base formulas φ, ψ , a trace $\rho = q_1, q_2, q_3, \ldots$ satisfies $\varphi \text{ U } \psi$ if $q_j \vDash \psi$ for some $j > 0$ and $q_i \vDash \varphi$ for all $i < j$

Example: trace:

```
x:  0  0  0  2  2  5  …
```

- satisfies $(x = 0)$ Until $(x = 2)$
- satisfies $(x < 5)$ Until $(x = 5)$

Note: If a trace satisfies φ Until ψ then it must also satisfy Eventually ψ

Nested Operators

- What does Next Always φ mean?
- Trace $\rho = q_1, q_2, q_3, \ldots$ satisfies Next Always φ if $q_j \vDash \varphi$ for all $j > 1$
- To formalize this, we have to define the relation
  $(\rho, j) \vDash \varphi$ (*trace* ρ *satisfies formula* φ *at position* j)
  - Same as *suffix* trace $q_j, q_{j+1}, q_{j+2}, \ldots$ starting at position j satisfies φ
  - $(\rho, j) \vDash \text{Next } \varphi$ if $(\rho, j+1) \vDash \varphi$
  - $(\rho, j) \vDash \text{Always } \varphi$ if $(\rho, k) \vDash \varphi$ for all positions $k \geq j$
  - $(\rho, j) \vDash \text{Eventually } \varphi$ if $(\rho, k) \vDash \varphi$ for some position $k \geq j$
  - $(\rho, j) \vDash \varphi \text{ U } \psi$ if there is a position $k \geq j$ such that
    $(\rho, i) \vDash \varphi$ for all $i = j \ldots k$ and $(\rho, k) \vDash \psi$
- Trace ρ *satisfies* φ iff $(\rho, 1) \vDash \varphi$

Multiple Eventualities

Example: Multi-agent system where multiple goals have to be satisfied

- Goal1: Robot 1 has finished its mission
- Goal2: Robot 2 has finished its mission

Spec: (Eventually Goal1) ∧ (Eventually Goal2)

- Trace ρ satisfies this spec if there are positions i, j such that $(\rho, i) \vDash$ Goal1 and $(\rho, j) \vDash$ Goal2
- No specific order specified in which goals are achieved

Spec: Eventually [Goal1 ∧ (Eventually Goal2)]

- Trace ρ satisfies this spec if there are positions i, j such that $i \leq j$ and $(\rho, i) \vDash$ Goal1 and $(\rho, j) \vDash$ Goal2

Spec: Eventually [Goal1 ∧ Next (Eventually Goal2)]

- Trace ρ satisfies this spec if there are positions i, j such that $i < j$ and $(\rho, i) \vDash$ Goal1 and $(\rho, j) \vDash$ Goal2

Recurrence and Persistence

Repeatedly φ = Always Eventually φ

- i.e., for every j, $(\rho, j) \vDash$ Eventually φ
- i.e., for every j, there is an $i \geq j$ such that $(\rho, i) \vDash \varphi$
- i.e., there are infinitely many positions where φ holds

Persistently φ = Eventually Always φ

- i.e., for some j, $(\rho, j) \vDash$ Always φ
- i.e., there is a j such that for all $i \geq j$, $(\rho, i) \vDash \varphi$
- i.e., formula φ becomes true eventually and stays true

The two patterns are logical duals:

a trace ρ satisfies Repeatedly φ iff it falsifies Persistently $\neg\varphi$

Examples

Trace:

x:	0	1	2	3	4	5	…
y:	0	1	0	1	0	1	…

Repeatedly $(y = 0)$

Persistently $(x \geq 10)$

Always [even(x) $\Rightarrow$ Next odd(x)]

Repeatedly prime(x)

Requirements-based Design

Given:

- Input/output interface of system C to be designed
- Model E of the environment
- LTL-formula φ over I/O variables and state variables of the environment model E

Design problem:

Fill in details of C so that every infinite execution of the composition of E and C satisfies φ

Applies to synchronous as well as asynchronous designs

Leader Election

Requirements refer to output variable $status_n$ of each node n

Liveness: Each node n eventually decides its status

$$\text{Eventually}\ (\ status_n = \text{leader} \lor status_n = \text{follower}\ )$$

Safety: For distinct nodes m, n, if m decides to be a leader at some point then n can never be a leader

$$\text{Eventually}\ (status_m = \text{leader}) \Rightarrow \text{Always}\ \neg(status_n = \text{leader})$$

Railroad Controller

Requirements refer to mode variables of trains and I/O variables (signals)

Safety: The two trains should not be on bridge simultaneously

Always $\neg(\text{mode}_W = \text{bridge} \wedge \text{mode}_E = \text{bridge})$

Liveness 1: West train gets on bridge repeatedly

Repeatedly $(\text{mode}_W = \text{bridge})$

Not a good spec (why?), no controller can satisfy this

Liveness 2: A waiting west train is eventually allowed to enter

Always [$(\text{mode}_W = \text{wait}) \Rightarrow$ Eventually $(\text{signal}_W = \text{green})$]

Note: LTL helps clarify ambiguities in English sentences

Formula is not satisfied by our controller (what is a counter-example?)

What if east train never leaves the bridge?

Railroad Controller

Liveness 2': Conditioned upon east train not staying on bridge forever

$\text{Repeatedly } \neg(\text{mode}_E = \text{bridge}) \Rightarrow$

$\text{Always[} (\text{mode}_W = \text{wait}) \Rightarrow \text{Eventually } (\text{signal}_W = \text{green}) \text{]}$

Does either of the two controllers in Chapter 3 satisfy this?

Liveness 3: If west train is waiting then eventually either it is allowed to enter bridge or east train is on bridge (implies absence of deadlocks)

$\text{Always [} (\text{mode}_W = \text{wait}) \Rightarrow$

$\text{Eventually } (\text{signal}_W = \text{green} \ \vee \text{mode}_E = \text{bridge}) \text{]}$

Writing precise requirements is challenging but crucial!

Credits

Notes based on Chapter 5 of

Principles of Cyber-Physical Systems
by Rajeev Alur
MIT Press, 2015