Principles of Cyber-Physical Systems

Liveness Requirements

Instructor: Lanshun Nie

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
- 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 || 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 in which the system may get stuck without achieving the goal
- Formalization:
  - Need to consider infinite executions (also called 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=6, y=0)
  - q_1 : (x=11, y=1)
- Base formula: Boolean-valued expression over V
  - even(x)
  - y=0 → even(x)
- Valuation q satisfies formula φ, written q |= φ, if q(φ) evaluates to 1
  - q_0 |= even(x)
  - q_0 |= y=0 → even(x)
  - q_1 does not satisfy even(x)
  - q_1 |= y=0 → even(x)

Traces

- Base formula states 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 context of system specification and verification:
  - V can be set of state variables, and then a trace corresponds to a possible infinite execution of the system
  - V can be set of input and output variables, and then a trace corresponds to an observed input/output behavior of system
  - V can include all of state, input, and output variables

LTL Basics

- Base formula states a property of a single valuation
- Trace: Infinite sequence of valuations
- LTL formula is evaluated with respect to a trace
- LTL formulas are built from base formulas using
  - Logical connectives (&, |, →, ~)
  - Temporal operators
- A trace $\rho = q_1, q_2, q_3, \ldots$ satisfies a base formula φ if $q_1 \models \varphi$

Always Operator

- Always φ means φ holds at all times
- Trace $\rho = q_1, q_2, q_3, \ldots$ satisfies Always φ if for all j, $q_j \models \varphi$
- Example trace

x:	0	1	2	3	4	5	…
y:	0	1	0	1	0	1	…

- Does not satisfy Always [even(x)]
- Satisfies Always [$y=0 \rightarrow$ even(x)]

- State property φ is an invariant of a transition system T iff every infinite execution of T satisfies Always φ

Eventually Operator

- Eventually φ means φ holds at some position (at least once)
- Trace $\rho = q_1, q_2, q_3, \ldots$ satisfies Eventually φ if for some j, $q_j \models \varphi$
- Example trace

x:	0	1	2	3	4	5	…
y:	0	1	0	1	0	1	…

- Satisfies Eventually [y = 1]
- Satisfies Eventually [x = 45]
- Does not satisfy Eventually [x=10 & y=1]

- Logical dual of Always: A trace satisfies Eventually φ if and only if it does not satisfy Always $\sim\varphi$

Next Operator

- Next φ means φ holds at "next" time
- Trace $\rho = q_1, q_2, q_3, \ldots$ satisfies Next φ if $q_2 \models \varphi$
- Example trace

x:	0	1	2	3	4	5 …
y:	0	1	0	1	0	1 …

- Satisfies Next [y = 1]
- Does not satisfy Next [x=2]

Until Operator

- φ Until ψ means ψ holds at some position and φ holds at all positions till then

- Trace $\rho = q_1, q_2, q_3, \ldots$ satisfies φ U ψ if for some j, $q_j \models \psi$ and for all $i < j$, $q_i \models \varphi$

- Example trace: x: 0 0 0 2 2 5 ...

- Satisfies (x=0) U (x=2)
- Satisfies (x<5) U (x=5)

- If a trace satisfies φ U 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 for all $j>=2$, $q_j \models \varphi$
- To formalize this, we have to define the relation $(\rho, j) \models \varphi$
  - 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
  - Trace ρ satisfies φ is same as $(\rho, 1) \models \varphi$
- $(\rho, j) \models$ Always φ if $(\rho, k) \models \varphi$ for every position $k >= j$
- $(\rho, j) \models$ Next φ if $(\rho, j+1) \models \varphi$
- $(\rho, j) \models$ Eventually φ if $(\rho, k) \models \varphi$ for some position $k >= j$
- $(\rho, j) \models \varphi \; U \; \psi$ if there exists position $k>=j$ such that $(\rho, k) \models \psi$ and for all positions i such that $j<=i<k$, $(\rho, i) \models \varphi$

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 exist positions i and j such that $(\rho, i) \models$ Goal1 and $(\rho, j) \models$ Goal2
  - No specific order specified in which goals are achieved
- Spec: Eventually [Goal1 & (Eventually Goal2)]
  - Trace ρ satisfies this spec if there exist positions i and j such that $i \le j$ and $(\rho, i) \models$ Goal1 and $(\rho, j) \models$ Goal2
- Spec: Eventually [Goal1 & Next (Eventually Goal2)]
  - Trace ρ satisfies this spec if there exist positions i and j such that $i < j$ and $(\rho, i) \models$ Goal1 and $(\rho, j) \models$ Goal2

Recurrence and Persistence

- Repeatedly φ = Always Eventually φ
  - For every position j, $(\rho,j) \models$ Eventually φ
  - For every j, there exists a position $i>=j$ such that $(\rho,i) \models \varphi$
  - There are infinitely many positions where φ holds

- Persistently φ = Eventually Always φ
  - For some position j, $(\rho,j) \models$ Always φ
  - There exists j such that for all positions $i>=j$, $(\rho,i) \models \varphi$
  - Formula φ becomes true eventually and stays true

- The two patterns are logical duals: A trace satisfies Repeatedly φ if and only if it does not satisfy Persistently $\sim \varphi$

Examples

- Example trace

x:	0	1	2	3	4	5 ...
y:	0	1	0	1	0	1 ...

Repeatedly (y=0)

Persistently (x >= 10)

Always [even(x) → 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 input/output variables and also state variables of the environment model E

- Design problem: Fill in details of C so that every infinite execution of the composite system satisfies the LTL-formula φ

- Applies to synchronous as well as asynchronous designs

Leader Election

- Requirements refer to output variable status of each node
- Liveness: Each node n eventually decides

  Eventually ($status_n$ = leader | $status_n$ = follower)
- Safety: For m!=n, if a node m decides to be a leader then node n cannot be a leader

  Eventually ($status_m$ = leader) → Always ($status_n$!= leader)

Railroad Controller

- Requirements refer to mode variables of trains and input/output variables (signals)
- Safety: Both trains should not be on bridge simultaneously
  - Always ~ ($mode_W$ = bridge & $mode_E$ = bridge)
- Liveness 1: West train gets on bridge repeatedly
  - Repeatedly ($mode_W$ = bridge)
  - Not a good spec (why?), no controller can satisfy this
- Liveness 2: A waiting west train is eventually allowed to enter
  - Always [($mode_W$ = wait) → Eventually ($signal_W$ = green)]
  - Note: LTL helps clarify ambiguities in English sentences
  - Not satisfied by our controller (what is a counter-example?)
  - What if east train never leaves the bridge??

Railroad Controller

- Liveness 3: Conditioned upon east train not staying on bridge forever
  - Repeatedly ($mode_E$!= bridge) → Always[($mode_W$ = wait) → Eventually ($signal_W$ = green)]
  - Do the two controllers in Chapter 3 satisfy this?

- Liveness 4: If west is waiting then eventually either it is allowed to enter or east is on bridge (this implies absence of deadlocks)
  - Always [($mode_W$ = wait) → Eventually ($signal_W$ = green | $mode_E$ = bridge)]

- Writing precise requirements is challenging (but important)

LTL Recap

- Syntax: Formulas built from
  - Base formulas: Boolean-valued expressions over typed variables
  - Logical connectives: AND, OR, NOT, IMPLIES …
  - Temporal Operators: Always, Eventually, Next, Until
- LTL formula is evaluated w.r.t. a trace ρ (infinite seq of valuations)
- Semantics defined by rules for the satisfaction relation
- A system satisfies LTL spec φ if every infinite execution satisfies φ
- Derived operators
  - Repeatedly (Always Eventually); Persistently (Eventually Always)
- Sample requirement: Every req is eventually granted

  Always [req=1 → Eventually (grant=1)]