CS 4387/5387
SOFTWARE V&V

LECTURE 9
FORMAL SPECIFICATIONS

Announcement

- Final Exam:
  - Tuesday May 14th 7:00 pm – 9:45 pm
Outline

- Formal Specifications Review
- Group Exercise (Game?)
- Quiz
- Spin
  - Overview
  - Installation
  - Reading Assignment

From Last Week

- $a \rightarrow b$
- Truth table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>$a \rightarrow b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>$\neg a \lor b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

If you study hard, you'll pass the exam;
1. There's nothing that can be asserted in the situation where you DON'T study
2. The only time this statement fails is if you study and you fail
LTL Syntax:

- Atoms: atomic propositions as in propositional logic

- Formulas (Q is an LTL formula):
  - \( Q ::= true \mid false \mid p \mid (\neg Q) \mid (Q \rightarrow Q) \mid (Q \land Q) \mid (Q \lor Q) \mid (X Q) \mid (F Q) \mid (G Q) \mid (Q \mathcal{U} Q) \mid (Q \mathcal{W} Q) \)

- \( p \in \text{ATOMS} \)
- \( X \) (Next)
- \( F \) (Future, \(<>)\)
- \( G \) (Global [\[]\])
- \( U \) (Until)
- \( W \) (Weak until)

Satisfaction-1

- Let \( \pi = s_1 \rightarrow s_2 \rightarrow \) be a path. \( \pi \) satisfies (\( \models \)) an LTL formula according to the following:
  1. \( \pi \models true \)
  2. \( \pi (\neg \models) false \)
  3. \( \pi \models p \iff p \in s_1 \)
  4. \( \pi \models Q_1 \land Q_2 \iff \pi \models Q_1 \) and \( \pi \models Q_2 \)
  5. \( \pi \models Q_1 \lor Q_2 \iff \pi \models Q_1 \) or \( \pi \models Q_2 \)
  6. \( \pi \models Q_1 \rightarrow Q_2 \iff \pi \models Q_2 \) whenever \( \pi \models Q_1 \)

These are the propositional rules; Note 3 says \( p \) must be true in the first state
### Satisfaction-2

7. \( \pi \models XQ \iff \pi^2 \models Q \)  
   true in next state

8. \( \pi \models G Q \iff \forall i \geq 1, \pi^i \models Q \)  
   always true

9. \( \pi \models FQ \iff \exists i \geq 1, \pi^i \models Q \)  
   true sometime in future

10. \( \pi \models Q_1 U Q_2 \iff \exists i \geq 1, \pi^i \models Q_2 \) and \( \forall 1 \leq j < i, \pi^j \models Q_1 \)

### Clause 10: Until

- **P U Q**
  - The first thing holds continuously until the second thing holds.
  - They don’t have to hold in the same state
  - The second thing has to occur

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPPQ__</td>
<td>TRUE</td>
</tr>
<tr>
<td><em>PQ</em>_</td>
<td>FALSE</td>
</tr>
<tr>
<td>Q_____</td>
<td>TRUE</td>
</tr>
<tr>
<td>PPPPPP</td>
<td>FALSE</td>
</tr>
<tr>
<td>QQPPQ_</td>
<td>TRUE</td>
</tr>
<tr>
<td>QQ_QQ</td>
<td>TRUE</td>
</tr>
<tr>
<td>QQ (PQ)Q</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
The future includes the present

The following are true in every model:

- $Gp \Rightarrow p$
- $p \Rightarrow (q \lor p)$
- $p \Rightarrow (Fp)$

Equivalence

- $((Fp) \land (Gq)) \Rightarrow (p \land r)$
- $(F(p \Rightarrow (G \lor r)) \lor (\neg q \lor p))$
- $(p \land (q \land r))$
- $(G(Fp)) \Rightarrow (F(q \lor s))$
Examples:

- The printer is always ready:
  - \( G \ p \)

- The printer will eventually be ready:
  - \( F \ p \)

- The printer will be ready until a job arrives:
  - \( p \ U \ j \)

- After a job arrives, sooner or later the printer will be ready
  - \( G(\ j \rightarrow F \ p) \)
  - Another suggestion: “\( F(\ j \rightarrow F \ p) \)”  Let’s try to validate this statement

Examples

The possible executions of this FSM is represented by this tree. Any given execution is a path from root to leaf.
Example

M, s0 |= p \&\& q (holds in first state)
M, s0 |= ! r (holds in first state)
M, s0 |= true (by definition)
M, s0 |= X r (all paths have r in 2nd state)
M, s2 |= G r TRUE (state 2)
F(!q \&\& r)
F G r If any path can get to !q \&\& r, then r is eventually continuously true on all states in the path
G F p p occurs infinitely often (Not true on all paths here)
M, s0 |= G F p -> G F r TRUE
M, s0 |= G F r -> G F p FALSE

Examples

- It is impossible to get to a state where we are started but not ready.
  - G ! (started \&\& ! ready)
- Every time a request is made, it will be serviced
  - G(requested -> F serviced)
- P is enabled infinitely often on every path
  - G F P_enabled
- Whatever happens, P will eventually become permanently deadlocked
  - F G P_deadlock
- If the process is enabled infinitely often, then it runs infinitely often
  - G F enabled -> G F running
- An elevator moving up at the second floor does not go down if it has passengers traveling to the 5th floor
  - G (floor2 \&\& directionup \&\& Button5) -> (directionup U Floor5)
Patterns: high-level abstractions that provide descriptions of common properties

Scopes: describe the extent of program execution over which a property holds

- **Patterns**
  - High-level abstractions that provide descriptions of common properties
  - Occurrence Patterns
  - Order Patterns

- **Examples**
  - **Existence:**
    - The temperature eventually reaches 100°C.
  - **Response:**
    - When a command expires, the train shall stop propulsion.
Scopes describe the extent of program execution over which the property holds.

Patterns: Examples

- Existence of P
- Q Precedes P
- Q Responds to P
Patterns/Scopes Description

- You are handed a description of every pattern and scope
- Your team is assigned a set of pattern/scope combinations
- Each team is tasked with the following:
  - Develop the LTL property for the 4 pattern/scopes you’ve been assigned
  - Verify the correctness of your specification using the Validator tool
  - Present to the class your team’s generated formula

Team Assignments

- You are to work with your teammates from last class
  - This is your team for the rest of the semester
Team 1 (Javier, Eduardo, Janelle)

- After_L/Absence
- Before_R/Existence
- Global/Response
- Before_R/Response

Team 2 (Madhu, Erik, Mikhail)

- Global/Existence
- After_L/Universality
- Between_L&R/Absence
- Between_L&R/Precedence
Team 3 (Edgar, Maria, Karina)
- Global/Universality
- Before_R/Absence
- After_L/Precedence
- Between_L&R/Existence

Team 4 (Marek, Josh, Luis)
- Before_R/Universality
- After_L/Existence
- Before_R/Precedence
- After_L/Response
Team 5 (Baltazar, Roberto, Juan)

- Global/Absence,
- Global/Precedence
- BetweenL&R/Universality
- BetweenL&R/Response

Competition Rules

- Each team has 4 properties to specify
  - Each property is worth 2.5 points for a total of 10 points
  - Each team will present one property at a time.
- Other teams will have five minutes to
  - Point defects in the specification by providing a counter example (for 1 point), and
  - Correct the specification (for the full 2.5 points)
- A team presenting a flawed specification will lose the full 2.5 points
VERIFICATION MODELS

Recap

- **Testing:**
  - Cannot test exhaustively
  - Can show presence but not absence of bugs
  - Expensive
  - Not effective on multiple threads

- **Formal Models:**
  - Formal specifications
  - Theorem proving
  - Model checking
Models

- Abstraction of system
- Used for analysis
  - Simplified system: analysis is tractable
  - Performed in design phase: before implementation
- Not new:
  - We’ve been using models for years:
    - Prototypes

Process of Engineering

- Requirements
- Design
- Analysis
- Implementation
Process of Engineering

- Requirements: Specification: Logic
- Design: Model: Prototype
- Analysis: Computation: Model Checking
- Implementation

Relevance of Model Checking

- 1968: software crisis
  - Complexity of software grows faster than our ability to control it
  - Large programs then were 100K lines of code
Relevance of Model Checking

- **1968: software crisis**
  - Complexity of software grows faster than our ability to control it
  - Large programs then were 100K lines of code
- **Today:**
  - Large programs are 10,000 times larger
  - Memory is 100,000 times larger
  - Speeds are 1,000,000 times faster

- **Software development: has it changed since 1968?**
Relevance of Model Checking

- 2012 software crisis: We cannot predictably produce reliable software
- This is the single most important unsolved problem in computer science

Standish Group Survey

- 1/3 of software projects are completed on time within budget
- More than 1/5 projects fail completely

![Bar chart showing project status by size]
Potential added value?

- Harmless and rare is not very important
- Catastrophic and rare is not adequately covered
- Model checking targets issues that have haunted developers for decades
- It is relevant, practical, and important

Feasibility of Model Checking

<table>
<thead>
<tr>
<th>Year</th>
<th>Memory Needed</th>
<th>Memory Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>1987</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>1995</td>
<td>1000</td>
<td>100000</td>
</tr>
<tr>
<td>2000</td>
<td>10000</td>
<td>1000000</td>
</tr>
</tbody>
</table>

- Model Checking targets issues that have haunted developers for decades.
Feasibility of Model Checking

Models

- **Abstraction**
  - Abstraction of reality
  - Less detail than the artifact being modeled
  - Details are selected on basis of relevance
  - Objective is to gain analytical power

- **Purpose**
  - Predict and explain
  - Too much or too little detail is not useful

- **Design Aid**
  - Often developed iteratively
  - Analysis often requires more than one model
How does it work?

- **System**  \( L(S) \) the set of possible behaviors of S

- **Property**  \( L(p) \) the set of valid behaviors

- **Prove that**  \( L(S) \subseteq L(p) \)  
  (Everything possible is valid)

- **Method**  Prove  \( L(S) \cap L(\neg p) = \emptyset \)
Types of errors we’re looking for

- Deadlocks, livelocks, starvation
- Race conditions
- Locking problems, priority problems
- Resource allocation errors
- Reliance on relative speeds
- Violations of known system bounds
- Specification incompleteness
- Specification redundancy
- Logic problems

Livelock

- A condition that occurs when two or more processes continually change their state in response to changes in the other processes.
- The result is that none of the processes will complete.
- An analogy is when two people meet in a hallway and each tries to step around the other but they end up swaying from side to side getting in each other’s way as they try to get out of the way.
Deadlock

- A condition that occurs when two processes are each waiting for the other to complete before proceeding.
- The result is that both processes hang.
- Deadlocks occur most commonly in multitasking and client/server environments.
- A deadlock is also called a *deadly embrace*.

Race Condition

- An undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence in order to be done correctly.
Starvation

- A multitasking-related problem, where a process is perpetually denied resources needed to complete its task.
- Deadlock occurs when two programs each hold resources the other needs to finish, and neither is willing to give them up.
- Starvation occurs when one program is unable to acquire the needed resource.
- Starvation is illustrated by Dijkstra’s dining philosophers problem.

Properties:

- **Reachability** property: some particular situation can be reached.
- **Safety property**: Under certain conditions, an event never occurs.
- **Liveness property**: Under certain conditions, some event will occur.
- **Deadlock freeness** property: a system can never be in a situation where no progress impossible
- A **fairness property** under certain conditions, an event will occur (or fail to occur) infinitely often.
Spin and SMV

- Two very popular model checkers
- SPIN
  - Software
  - Linear temporal logic
- SMV
  - Hardware
  - Computation Tree Logic
- Lots of others

Next Week

- Writing Models in PROMELA
- Reading assignment on blackboard
  - Quiz at the beginning of class (That’s a promise)
- Have the spin model checker installed on your personal laptop
- http://spinroot.com