18-642 Recitation
Week #10
November 1, 2018
Updates

• Homework:
  – Last week homeworks graded on canvas
  – Next week homework due Wednesday night
    • Project workload has been INCREASING
    • Homework workload is DECREASING (only HW #30, 38, 40)

• Homework grading
  – Points are mostly for effort
  – Read comments on canvas, even if you got full points
  – Full points does not mean you got the right answer
    • We’ll try to cover some common issues in recitation
    • If you’re not sure – ask!
Updates

• Projects:
  – Project 7 graded on canvas
    • If you handed in late, it might take a while longer
  – Project 8 due tonight
  – Project 9 is live

• Don’t forget the weekly survey!
Today

• Project 8 questions?
• Project 9

• Be sure to fill out weekly surveys!
Project #8 Questions?

- Peer reviews on code
- Unit Testing
- Check, but do not fix warnings
Project 9

- Fix compiler warnings
- Fix any failing unit tests
- Peer review of unit tests
- First step for monitoring/invariants

- Preview of Project 10:
  - More monitoring/invariants
Compiler Warning Flags

• Same as in Project 8

```bash
# Warning flags for Projects 8,9,10
target_compile_options(ece642rtle_student PUBLIC
    "-Werror" # Do not comment this out after Project 8!
    "-Wextra"
    "-Wall"
    "-Wfloat-equal"
    "-Wconversion"
    "-Wparentheses"
    "-pedantic"
    "-Wunused-parameter"
    "-Wunused-variable"
    "-Wreturn-type"
    "-Wunused-function"
    "-Wredundant-decls"
    "-Wreturn-type"
    "-Wredundant-decls"
    "-Wunused-value"
    "-Wswitch-default"
    "-Wunused-initialized" "-O1" "-Winit-self"
)
```
Compilation Notes

- Remember: Makefiles only rebuild source files if they have changed (or if they haven’t successfully compiled before)
- If you don’t change any files, warnings won’t appear on second compile
- Our solution: keep the –Werror flag
  - Turns all warnings into errors
  - Compilation won’t succeed until all warnings are removed
- If you disable the –Werror flag, remember to re-enable it and check build for warnings
- Turn in Project 9 code with all warning flags, especially –Werror, enabled
  - Make sure you compile it with –Werror before turning it in
Unit Tests

• In Project 8, you wrote unit tests to achieve 100% transition, 100% branch, and sufficient data coverage
• For Project 9, you must maintain this coverage and have your tests pass
Unit Test Peer Review

- Groups assigned on canvas
- Same organization/etiquette as previous projects
- Review testing coverage and traceability
  - Have your state chart printed or on a tablet screen
- Create a unit test peer review checklist as before
  - For each transition: check that the test exists and has a commented annotation in code, and that the test is correct
  - Check testing code style, but test coverage is the main emphasis
  - Check whether pass/fail criteria are correct
Runtime monitors

- In general: some sort of external entity that plugs into a system’s interface to monitor system output and state
- For ece642rtle: a separate ROS node that listens to messages sent by the turtle system
- You are given an interface for handling message interrupts
- By paying attention to the sequence of messages, can infer state and detect improper behavior
Logging Monitor

- Example provided to you to show you how message interrupt handling works
- Just prints every message it sees
- Doesn’t record state or check invariants

```c
#include "monitor_interface.h"

void tickInterrupt(bool solved) {
    ROS_INFO("One call to moveTurtle finished. Maze solved: %s", solved ? "true" : "false");
}

void poseInterrupt(int x, int y) {
    ROS_INFO("'Pose' was sent. Data: x = %d, y=%d", x, y);
}

void orientationInterrupt(Orientation o) {
    std::string o_str;
    switch(o) {
    case NORTH:
        o_str = "NORTH";
        break;
    case WEST:
        o_str = "WEST";
        break;
    case SOUTH:
        o_str = "SOUTH";
        break;
    case EAST:
        o_str = "EAST";
        break;
    default:
        o_str = "ERROR";
        break;
    }
    ROS_INFO("'Orientation' was sent. Data: %s", o_str);
}

void visitInterrupt(int visits) {
    ROS_INFO("'Visits' was sent. Data: %d", visits);
}````
Invariants

• A condition that is always true during proper operation
• A monitor can check for invariant violations by keeping track of the state of the system
Step Monitor

• Invariant: turtle shall not move more than one square at a time
• Example provided to you to show how monitor can log state to check invariants
• Uses ROS_WARN to indicate an invariant violation

```c
/*
 * Returns absolute value of x
 * WARNING: unsafe for edge-case values
 */
inline int abs(int x) {
  return x < 0 ? -1*x : x;
}

/*
 * Whenever the turtle moves, compare the current location
 * to the previous location and throw an invariant violation
 * if the locations differ by more than 1 in Manhattan Distance.
 */
void poseInterrupt(int x, int y) {
  // Print pose info
  // Last conditional makes sure that if suppress_double_visits is
  // true, that the same pose isn't printed twice
  if ((!suppress_double_visits || !moved) ||
      (last_pose.x != x || last_pose.y != y)) {
    ROS_INFO("Pose was sent. Data: x = %d, y=%d", x, y);
  }

  // Check that the turtle has moved before and that the Manhattan
  // distance between the positions does not exceed 1
  if (moved && (abs(last_pose.x - x) + abs(last_pose.y - y)) > 1) {
    ROS_WARN("VIOLATION: Difference between last coordinate (%d,%d) and
     current coordinate (%d,%d) is more than 1 square!", last_pose.x, last_pose.y, x, y);
  }

  // store last Pose in memory
  last_pose.x = x;
  last_pose.y = y;

  // Update this flag the first time the turtle moves
  if (!moved) {
    moved = true;
  }
```
Turn monitor

- Invariant: turtle shall not turn more than 90 degrees at a time
- You must write this invariant
- Project page gives some hints
- Run it with your turtle
  - If invariant is violated: is it because of your turtle code, or a bug in the monitor?
  - You do not have to fix invariant violations for Project 9
Project 9 summary

- Fix warnings
- Fix failing unit tests
- Peer review unit tests (not the turtle code itself)
- Invariant monitor for 90 degree turn limit
- Lots of choices to manage your time:
  - Unit tests compile independent of ROS project, so you can fix warnings first OR fix unit tests first
  - Write monitor at any time
- Make sure what you turn in compiles successfully and that the documentation is up-to-date
Looking ahead to Project 10

• Solve several new mazes (acceptance tests)
• Update documentation/traceability/unit tests
• Write more invariants
• Peer reviews of the invariant monitors
Project Questions?
Scary Failure Mode Stories & Your Estimated SIL Rating

- 26-1-1: jet aircraft engine
- 26-1-2: cell phone
- 26-1-3: elevator
- 26-1-4: escalator
- 26-1-5: automatic blood pressure meter
- 26-1-6: microwave oven
- 26-1-7: pressure cooker (instant pot)
- 26-1-8: grocery store freezer (frozen food section)
- 26-1-9: automatic bathroom water faucet
- 26-1-10: household thermostat (room temperature control)