Security Engineering
3rd Problem Set

Prof Stefan Lucks, Nathalie Jolanthe Dittrich

`<first>.<middle>.<lastname>(at)uni-weimar.de`

Bauhaus-Universität Weimar

23 Nov 2018

Section 1

Organization

Mini projects

If you are presenting a mini project, please submit it till Thursday 13:30 before the Problem Session.

Section 2

Miniprojects

Recap

- Get familiar with the basics of Ada: ✓
- Get familiar with unit testing: ✓
- Get familiar with at least one testing tool: ✓
- Organize sources and testing code: ✓

Section 3

4th Problem Set

4th Problem Set: Next Steps

- Reflect the quality of your tests
- The formal backgrounds of how-to-prove the correctness of program code
- Proofing the correctness of your code

How to Measure Test Quality?

- Analyzing covered lines of source code
- Analyzing covered branches
- Rule-of-thumb:
  - All lines covered is the wish (loops?)
  - $> 95\%$ of lines are desirable

`gcov` and `lcov`

- `gcov`
  - Many of you already know (comes with `gcc`)
    `gcc.gnu.org/onlinedocs/gcc/Gcov.html`
  - Adds invocation points to each line of code
- `lcov`
  - Set of Linux tools to derive human-readable coverage results
  - `genhtml` to create HTML report

gcov

1. Hint: Clean all `.gcno` results from previous runs
2. Add `–coverage` switch to `gnatmake`
   (or alternatively `–f –cargs –fprofile-arcs –ftest-coverage –largs –fprofile-arcs`)
3. Run your program once
4. Run `gcov` to generate coverage results (`*.gcno`)
   (there was a reason why we used `lcov –gcov-tool gcov` instead)

```
gcov: makefolders clean
  gnatmake -P$(PROJECT_FILE) -f -cargs -coverage -largs -coverage
  chmod +x $(PROGRAM_DIR)/$(PROGRAM_NAME)
  $(PROGRAM_DIR)/$(PROGRAM_NAME)
  lcov --gcov-tool $(GCOV_PROJECT_PATH) --directory $(PROGRAM_DIR) \
       --capture --output-file $(LCOV_FILE)
```

lcov

1. Run `lcov` with the `-directory` switch
2. Run `genhtml`
3. Hint: Remove uninteresting packages with `lcov -r` before running `genhtml`

```
gcov:
  # ...
  lcov --gcov-tool $(GCOV_PROJECT_PATH) --directory $(PROGRAM_DIR) \
       --capture --output-file $(LCOV_FILE)

lcov:
  lcov -r $(LCOV_FILE) "*.ads" -o $(LCOV_FILE)
  lcov -r $(LCOV_FILE) "aunit*.adb" -o $(LCOV_FILE)
  lcov -r $(LCOV_FILE) "a-*.adb" -o $(LCOV_FILE)
  lcov -r $(LCOV_FILE) "harness/*" -o $(LCOV_FILE)
  lcov -r $(LCOV_FILE) "tests/*" -o $(LCOV_FILE)
  lcov -r $(LCOV_FILE) "bin/*" -o $(LCOV_FILE)
  genhtml $(LCOV_FILE) --output-directory $(GENHTML_OUTPUT_DIR) --legend -s
```

Section 4

Hoare Logic

Hoare Logic

- Grammar for expressions, assignments, conditions, etc.
- From 1969 but very close to today programming
- Set of rules to transform expressions
- Task: Derive post-conditions from pre-conditions and transformation rules

```
function Exponentiate(N: Natural; B: Natural) is
  {Pre-Condition := N >= 0, B >= 0}
  X: Natural;
  Y: Natural;
begin
  X := 0;
  Y := 1;
  while X /= N loop
   X := X + 1;
   Y := Y * B;
  end loop;
  {Post-Condition := Y = B^N}
  return Y;
end Exponentiate;
```

Program Correctness

- A program is **P** correct $\Leftrightarrow$ for all possible inputs that fulfill the pre-conditions, **P** always fulfills all post-conditions.
- **Partial correctness:**
  - **P** is correct *if* it terminates.
- **Total correctness:**
  - **P** is correct *and* always terminates.

Loops

- How can one prove that a loop terminates?

⇒ Loop variant `V`:

  - Decreases with every iteration
  - Loop is exited when `V <= 0`
  - Trivial for `for` loops:
    `V := Range'Last - I`

```
{N > 0, B > 0}
X := 0;
Y := 1;
{X = 0, Y = 1}
{Y = B^X} (Loop Invariant)
while X /= N loop
 {X /= N}        (Condition)

 X := X + 1;
 {X = X'Old + 1} (Assignment)

 Y := Y * B;
 {Y = Y'Old * B} (Assignment)
 {Y = B^X} (Loop Invariant)
end loop;
{X = N}   (Inverse Condition)
{Y = B^X} (Loop Invariant)
{Y = B^N} (Implication)
```

Loops

- How to prove that loops lead to correct post-conditions?

$\Rightarrow$ Loop invariant `IV` $:=$ $Y = B^X$:

  - Must be valid before the loop
  - Must be valid at each loop iteration

```
{N > 0, B > 0}
X := 0;
Y := 1;
{X = 0, Y = 1}
{Y = B^X} (Loop Invariant)
while X /= N loop
 {X /= N}        (Condition)

 X := X + 1;
 {X = X'Old + 1} (Assignment)

 Y := Y * B;
 {Y = Y'Old * B} (Assignment)
 {Y = B^X} (Loop Invariant)
end loop;
{X = N}   (Inverse Condition)
{Y = B^X} (Loop Invariant)
{Y = B^N} (Implication)
```

Questions?