Security Engineering
4th Problem Set

Prof Stefan Lucks, Nathalie Jolanthe Dittrich
<first>.<middle>.<lastname>(at)uni-weimar.de

Bauhaus-Universität Weimar

Dec 07, 2018

Section 1

4th Problem Set

4th Problem Set (cont'd)

- `gcov/lcov`
  - "Measure" the quality of your tests

4th Problem Set (cont'd)

- `gcov/lcov`
  - "Measure" the quality of your tests
- Data and Control Flow
  - Repetition (for formal verification)

4th Problem Set (cont'd)

- `gcov/lcov`
  - "Measure" the quality of your tests
- Data and Control Flow
  - Repetition (for formal verification)
- Get familiar with Hoare logic
  - Preparation for formal verification with SPARK =)

4th Problem Set (cont'd)

- `gcov/lcov`
  - "Measure" the quality of your tests
- Data and Control Flow
  - Repetition (for formal verification)
- Get familiar with Hoare logic
  - Preparation for formal verification with SPARK =)
- Access types and data structures

Section 2

`gcov` and `lcov`

Mini Project

Section 3

Hoare Logic I

Mini Project

Section 4

Hoare Logic II

Mini Project

Rules

Recap

- Rule 1: Empty Statement

$$\frac{\text{True}}{\{P\}\text{null};\{P\}}$$

- Rule 2: Assignment

$$\frac{\text{True}}{\{P[\text{Expr} \rightarrow v]\}v := \text{Expr};\{P\}}$$

- Rule 3: Composition

$$\frac{\{P\}S_1\{R\},\{R\}S_2\{Q\}}{\{P\}S_1;S_2;\{Q\}}$$

Rules

Recap

- Rule 4: If-Then-Else

$$\frac{\{P, B\}S_1\{Q\}, \{P, \neg B\}S_2\{Q\}}{\{P\} \text{ if } B \text{ then } S_1; \text{ else } S_2; \text{ end if;} \{Q\}}$$

- Rule 5: While

$$\frac{\{P, B\}S\{P\}}{\{P\} \text{ while } B \text{ loop } S; \text{ end loop;} \{P, \neg B\}}$$

- Rule 6: Conclusion

$$\frac{P \implies P', \{P'\}S\{Q'\}, Q' \implies Q}{\{P\}S\{Q\}}$$

Section 5

Access Types and Data Structures

Binary Tree

Mini Project

Section 6

5th Problem Set

Recap: What You Already Learnt

- Get to know basics of Ada: ✓
- Get familiar with at least one testing tool: ✓
- Learn how-to do white-box and black-box testing: ✓
- Practicing test coverage: ✓
- Get familiar with formal verification logic: ✓
- Get familiar with access types and data structures: ✓

Up to come:

- Get familiar with SPARK 2014
- Parallel programming with Tasks in Ada
- Scheduling algorithms

Formal verification with SPARK 2014

- Practice annotations for proving correct data flow
- Use your pre and post-conditions from Ada 2012
- Write shortened helper functions
- Prove small programs

Get familiar with `gnatprove`

- The usage is simple:
  `gnatprove -P$(projectfile)`
- The project file needs `-mode=[flow|prove|check|all]`
  `-report=[all|fail]`
- You may need a config `.adc` (or `-gnato13` for `gnatmake`)
- You need `pragma Spark_Mode(On)` to activate proving

An Example Project File

```
project Ps5_P2_Prove is
   type Modes is ("Compile", "Analyze");
   Mode : Modes := External ("MODE", "Compile");

   for Main use ("main.adb");
   for Source_Dirs use ("src");
   for Object_Dir use "bin";

   package Compiler is
      case Mode is
         when "Compile" =>
            for Switches ("Ada") use ("-g", "-gnato", "-gnatwa", "-gnatQ",
               "-gnat12");
         when "Analyze" =>
            for Switches ("Ada") use ("-g", "-j2", "-gnato", "-gnatwa",
               "-gnatQ", "-gnat12", "-gnato13");
      end case;
   end Compiler;

   package Prove is
       for Switches use ("--report=all", "--proof=progressive",
                         "--warnings=continue", "--timeout=5", "--mode=all");
   end Prove;

   package Builder is
      for Default_Switches ("ada") use ();
   end Builder;

end Ps5_P2_Prove;
```

Hints for Working with `gnatprove`

```
% gnatprove -Pps5_p2_prove.gpr
Phase 1 of 3: frame condition computation ...
Phase 2 of 3: analysis and translation to intermediate language ...
Phase 3 of 3: generation and proof of VCs ...
warning: no bodies have been analyzed by GNATprove
enable analysis of a body using SPARK_Mode
```

No errors does **not** always mean you're done!
The above means nothing has been tested!

1. Implement your package
2. Write tests to verify the correctness of your code
3. Copy your tests to a main (AUnit may give conflicts)
4. Use your main for proving with `gnatprove`
5. Check the output `gnatprove.out`

More Hints for Working with `gnatprove`

- If in doubt: You are wrong and `gnatprove` is right
  - `gnatprove` often sees edge cases we overlooked...
- Find appropriate `Loop_Invariant` s
- Add range checks! If necessary, include edge cases in your pre-/postconditions in an `(if <edge case>) or (not <edge case> and <post-condition>)`
- Use helper functions to manage the complexity
- Inline helper functions

```
function Contains(A: Natural_Array; Value: Natural;
  From: Natural; To: Natural) return Boolean is
  (for some I in From..To =>
    (if I in A'Range then A(I) = Value else False));
```

- Go step by step forwards
- Be patient... it's experimental

Use Your Resources Well

- The lecture slides
- The AdaCore introduction to SPARK 2014
- Read about `Depends` and `Globals` for proving `flow`:

```
procedure Initialize(Num_Voters: Natural) with
  Global  => (
      Input    => (Num_Votes_Made), -- Globals that are only read
      In_Out   => (Num_Votes_Made), -- Globals that are read and written
      Proof_In => (Num_Votes_Made), -- Globals that are only read,
      -- and only listed for stopping gnatprove from nagging
      Output   => (Num_Votes_Made)  -- Globals that are only written
  ),
  Depends => (
      X => (<The variables that X depends on>),
      Y => null, -- Y depends on nothing
      ...
  ),
  Pre     => (...),
```

- Read about `pragma Loop_Invariant(...))`
- Read about `Contract_Cases`
  - It may simplify life when you have many cases in post conditions

Questions?