Introduction to Program Reasoning

19CSE205 : PROGRAM REASONING

Dr. Swaminathan J

Assistant Professor

Department of Computer Science and Engineering

AMRITA VISHWA VIDYAPEETHAM

Jul - Dec 2020

Contents

What is Program Reasoning?

The task of reasoning about the correctness of a program, for a given specification, either through manual or automated means.

- The goal is to identify the presence of errors or prove their absence.

- Static approaches ⇒ Based on source code
  - Code inspection
  - Peer review
  - Static analysis
  - Formal verification

- Dynamic approaches ⇒ Based on program execution
  - Testing
  - Debugging
  - Tracing
  - Instrumentation

Code inspection & Peer review

A formal review carried out by self, peer and/or group to evaluate the quality of code. Usually a manual activity. Errors are categorized based on the severity of their impact.

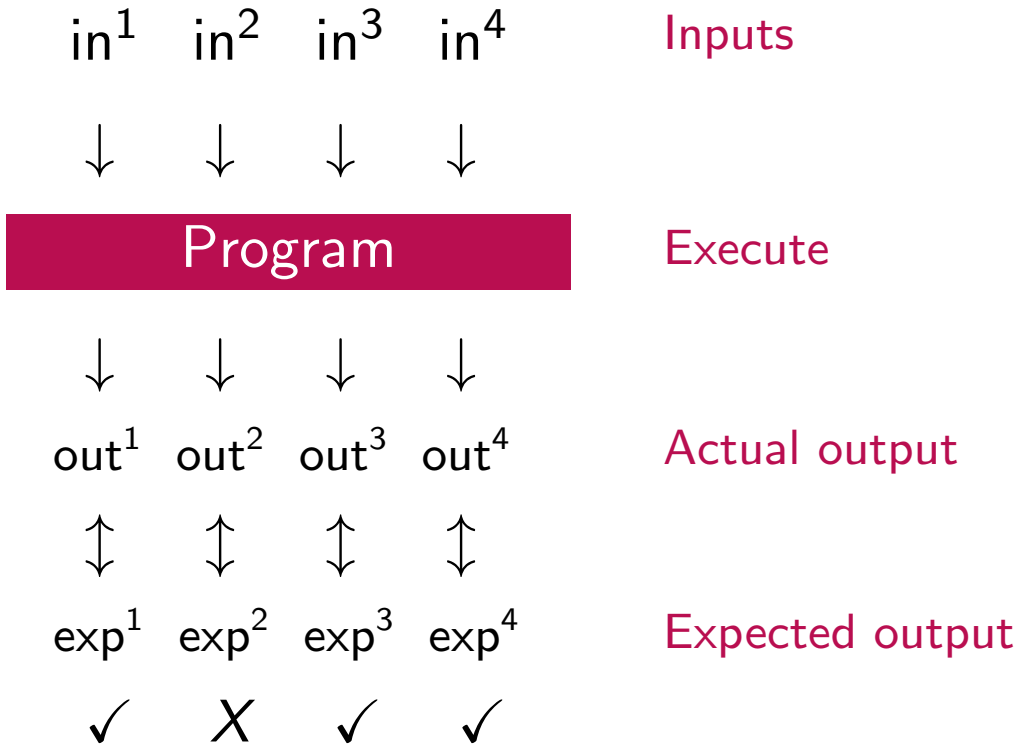
- Static approaches
  - Code inspection
  - Peer review
  - Static analysis
  - Formal verification
- Dynamic approaches
  - Testing
  - Debugging
  - Tracing
  - Instrumentation

Good quality code is

- Modular
- Readable
- Correct
- Adheres to standards
- ...

Execution of the program with various (preferably all possible) inputs and checking the output. Testing can be either manual or automated.

- Static approaches
  - Code inspection ✓
  - Peer review ✓
  - Static analysis
  - Formal verification
- Dynamic approaches
  - Testing
  - Debugging
  - Tracing
  - Instrumentation



Debugging

The process of locating errors in the code and fix them. It is a manual activity. Debuggers are integral part of almost all IDEs.

- Static approaches
  - Code inspection ✓
  - Peer review ✓
  - Static analysis
  - Formal verification
- Dynamic approaches
  - Testing ✓
  - Debugging
  - Tracing
  - Instrumentation

Debuggers allow users to

- Pause execution by setting breakpoints
- Inspect program state and modify them
- Step into/out of/skip functions

Tracing

Tracing is the process of inserting print statements to the code to trace the program flow. It is usually a manual activity.

- Static approaches
  - Code inspection ✓
  - Peer review ✓
  - Static analysis
  - Formal verification
- Dynamic approaches
  - Testing ✓
  - Debugging ✓
  - Tracing
  - Instrumentation

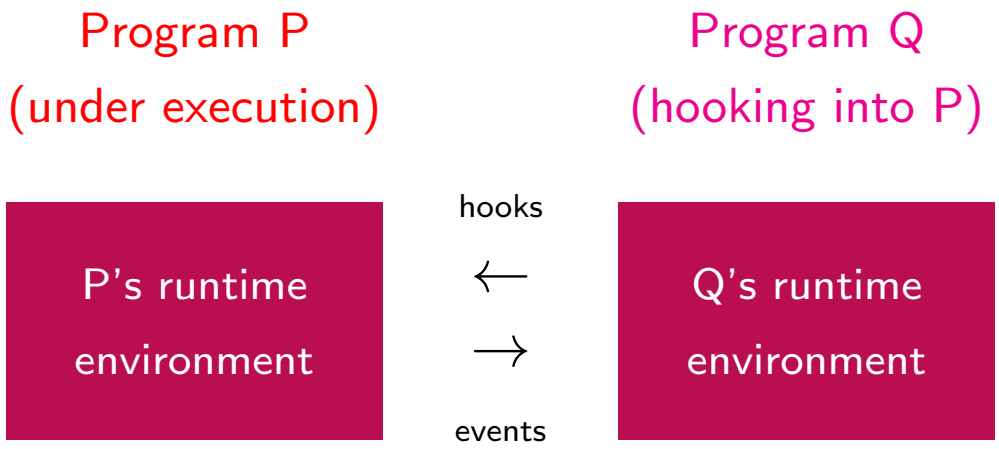
Tracing a factorial program

```
int factorial(int n) {
    int fact = 1;
    printf("%d ",fact);
    for (int i=2; i<n; i++)
        fact = fact * i;
        printf("%d ",fact);
    return fact;
}
int main() {
    int result = factorial(6);
}
```

1 2 6 24 120

Instrumentation is automatic injection of print/log statements to source or binary code. An alternate method is to hook into the program execution (much the same way debugger does), which then spits out the runtime events by pause-spit-resume mechanism (thus providing event trace).

- Static approaches
  - Code inspection ✓
  - Peer review ✓
  - Static analysis
  - Formal verification
- Dynamic approaches
  - Testing ✓
  - Debugging ✓
  - Tracing ✓
  - Instrumentation



Static analysis

Static analysis is an automated way to analyze the source code. The source code is first converted to a tree or graph form and analysis is carried out by traversing through the structure.

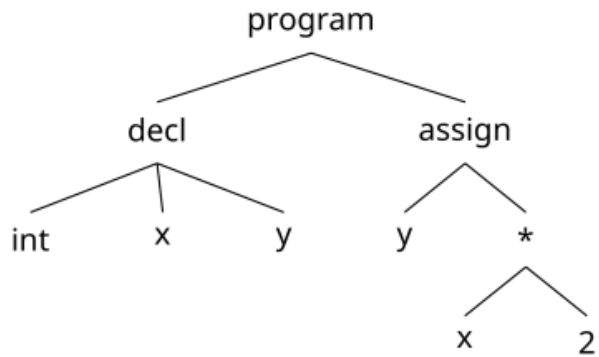
- Static approaches
  - Code inspection ✓
  - Peer review ✓
  - Static analysis
  - Formal verification
- Dynamic approaches
  - Testing ✓
  - Debugging ✓
  - Tracing ✓
  - Instrumentation ✓

Sample program

```
int x, y;
y = x * 2;
```

The initial value of x is not set

Abstract Syntax Tree



Symbol table

var	type	value
x	int	?
y	int	?

There are so many representations and several analysis techniques!

Formal Verification

The program is turned into logical formulae or a model. User states the correctness criteria. Theorem provers / SMT solvers are then used to prove that correctness specifications are met.

- Static approaches
  - Code inspection ✓
  - Peer review ✓
  - Static analysis ✓
  - Formal verification
- Dynamic approaches
  - Testing ✓
  - Debugging ✓
  - Tracing ✓
  - Instrumentation ✓

Verification in Frama-C

```
/*@ ensures \result >= a
          && \result >= b;
*/
int max(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```

Unlike other methods discussed earlier, which seek to identify errors, formal verification seeks to prove the absence of errors.

Terms and their meanings

- Static: Based on source (or executable) code
- Dynamic: Based on execution of the program
- Manual: Activity carried out by a human
- Automated: Activity performed by a program
- Semi-automated: Partly automated, human intervention necessary
- Code inspection: Examining source code to identify errors
- Peer review: A peer inspects the source code
- Static analyser: A program that analyzes the code and reports warnings and potential errors
- Program verifier: A program that takes source code and correctness criteria from user to ascertain if they will be met
- Testing: Execution of the program with different inputs and check if the actual output deviates from the expected
- Debugging: Interrupt the execution to examine the state in order to determine the cause of an error
- Tracing: Insert print statements in the program to trace errors
- Instrumentation: A program that inserts prints statements automatically during the execution