Formal Methods

Dr. Mark C. Paulk

SE 4367 – Software Testing, Verification, Validation, and Quality Assurance

Three Mechanisms for V&V

Testing

- **"test" can be a generic word – see Weinberg's Perfect Software and Other Illusions About Testing**
  - **does (not) meet need**
  - **does (not) meet all requirements**
  - **(un)acceptable costs or constraints**
  - **grossly unreliable**
  - **poor performance**
- **by testing, we typically assume dynamic execution of a program**
  - **black box, white box, unit, integration, system, regression, etc., testing**

Formal methods

- formal specifications
- model / property checking
- proofs of correctness
  - Correctness attempts to establish the program is error-free; testing attempts to find if there are any errors in the program.

Peer reviews

- A review of a software work product, following defined procedures, by peers of the producers of the product for the purpose of identifying defects and improvements. (Software CMM)
  - managers are not peers…
  - customers are not peers…

Formal Method

A technique for expressing requirements in a manner that allows the requirements to be studied mathematically.

Formal methods allow sets of requirements to be examined for completeness, consistency, and equivalency to another requirement set.

Formal methods result in formal specifications.

- **EIA 731**

Levels of Formal Method

Level 0

- **Formal specification may be undertaken and then a program developed from this informally.**

Level 1

- **Formal development and formal verification may be used to produce a program in a more formal manner.**
  - **proofs of properties or refinement from the specification to a program may be undertaken**
  - **may be most appropriate in high-integrity systems involving safety or security**

Level 2

- **Theorem provers may be used to undertake fully formal machine-checked proofs.**

Formal Semantics

Denotational semantics

- **meaning of a system is expressed in the mathematical theory of domains**

Operational semantics

- **meaning of a system is expressed as a sequence of actions of a (presumably) simpler computational model**

Axiomatic semantics

- **meaning of the system is expressed in terms of preconditions and postconditions which are true before and after the system performs a task, respectively**

Automated Proofs

Automated theorem proving

- **a system attempts to produce a formal proof from scratch, given**
  - **a description of the system**
  - **a set of logical axioms**
  - **a set of inference rules**

Model checking

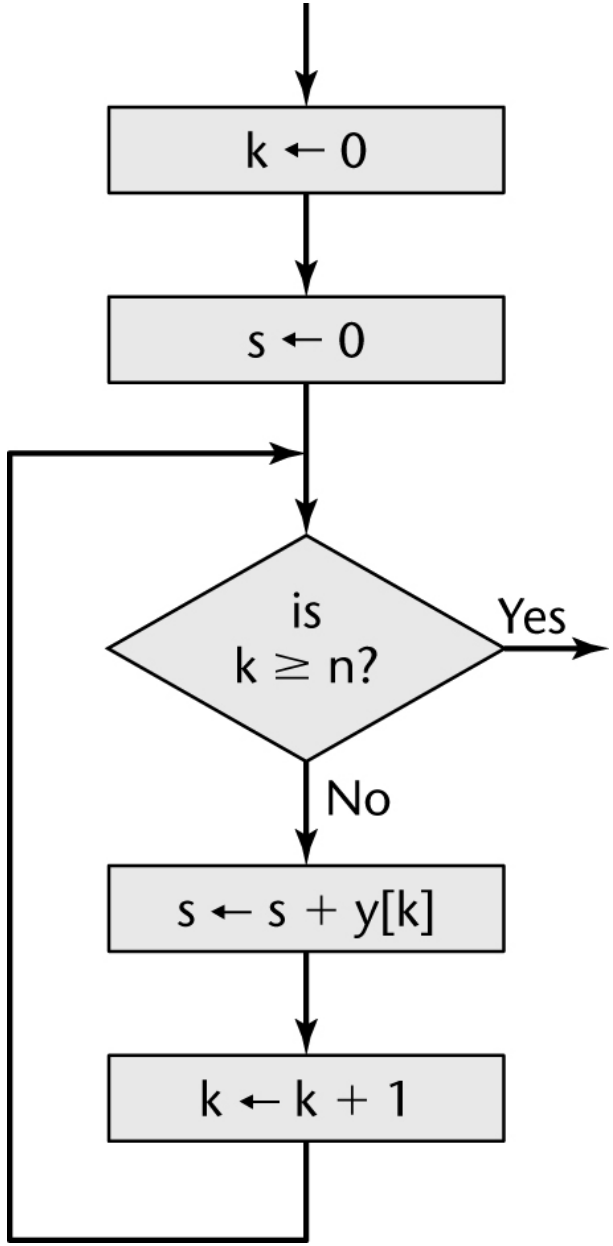
- **a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution**

Example of a Correctness Proof
(Schach)

The code segment to be proven correct: after execution, the variable s will contain the sum of the n elements of array y.

```
int k, s;
int y[n];
k = 0;
s = 0;
while (k < n)
{
    s = s + y[k];
    k = k + 1;
}
```

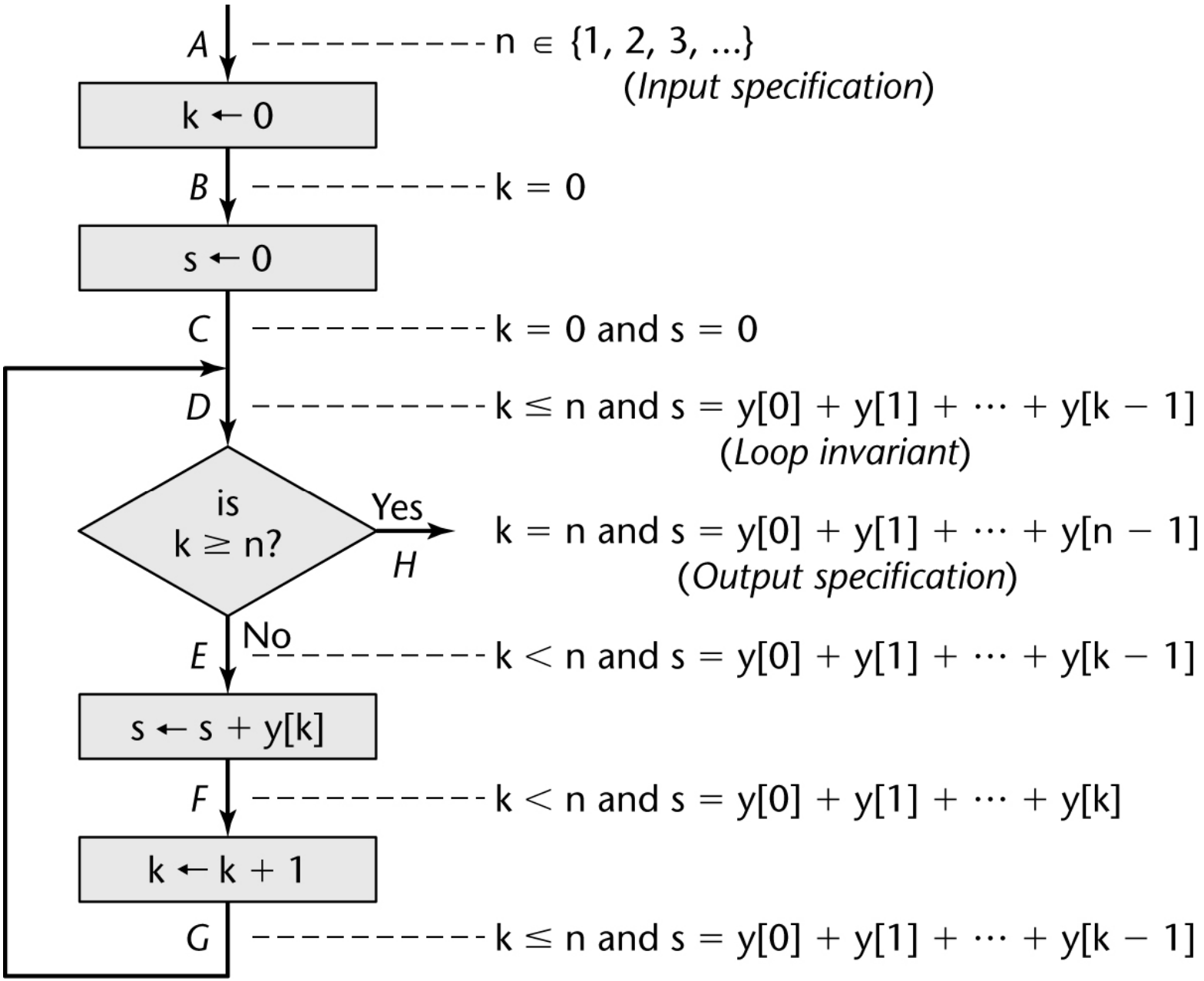
A flowchart equivalent of the code segment



Add to the flowchart

- **input specification**
- **output specification**
- **loop invariant**
- **assertions**

An informal proof (using induction) appears in Section 6.5.1 of Schach.


A
n ∈ {1, 2, 3, ...}
(Input specification)
k ← 0
B
k = 0
s ← 0
C
k = 0 and s = 0
D
k ≤ n and s = y[0] + y[1] + ⋯ + y[k − 1]
(Loop invariant)
is
k ≥ n?
Yes
H
k = n and s = y[0] + y[1] + ⋯ + y[n − 1]
(Output specification)
No
E
k < n and s = y[0] + y[1] + ⋯ + y[k − 1]
s ← s + y[k]
F
k < n and s = y[0] + y[1] + ⋯ + y[k]
k ← k + 1
G
k ≤ n and s = y[0] + y[1] + ⋯ + y[k − 1]

Specification for a Sort

Input spec

- **p: array of n integers, n>0**

Output spec

- **q: array of n integers, n>0 such that**
- **q[0] ≤ q[1] ≤ … q[n-1]**

```
void tricksort (int p[], int q[])
{
    for (int i, i<n, i++)
        q[i] = 0;
}
```

A Better Specification

Input spec

- **p: array of n integers, n>0**

Output spec

- **q: array of n integers, n>0 such that**
- **q[0] ≤ q[1] ≤ … q[n-1]**
- **the elements of array q are a permutation of the elements of array p, which are unchanged**

Correctness

A product is correct if it satisfies its specifications.

- **Is a correctness proof an alternative to execution-based testing?**

***Not* necessary**

- **C++ compiler may have an erroneous error message but be superior in all other ways to the alternatives**

***Not* sufficient**

- `trickSort` **satisfies the specification but not the intent**

Correctness + Testing

Correctness is established via mathematical proofs of programs.

Proofs are precise but subject to human fallibility.

"Beware of bugs in the above code; I have only proved it correct, not tried it."

- ***Donald Knuth, 1977***

"Program testing can be used to show the presence of bugs, but never to show their absence!"

- ***Edsger Dijkstra, "Notes on Structured Programming," 1970***

Correctness Proof Case Study
Naur Text-Processing Problem

Given a text consisting of words separated by blank or by newline characters

Convert it to line-by-line form in accordance with the following rules

- **line breaks must be made only where the given text contains a blank or newline**
- **each line is filled as far as possible, as long as**
- **no line will contain more than maxpos characters**

Timeline for the Naur Proof

1969 – Naur paper

- **Naur constructed a procedure and informally proved its correctness**
  - **about 25 lines of Algol 60**

1970 – Leavenworth in *Computing Reviews* found 1 fault

- **the first word of the first line is preceded by a blank unless the first word is exactly maxpos characters long**

1971 – London found 3 more faults

- **including that the procedure does not terminate unless a word longer than maxpos characters is encountered**

London formally proved his corrected specification was correct

1975 – Goodenough and Gerhart found 3 more faults

- including that the last word will not be output unless it is followed by a blank or newline

- **Goodenough and Gerhart produced a new specification, about four times longer than Naur's**

Note that four of the seven faults would have been detected if the procedure had been tested using the data in Naur's original paper.

1985 – Meyer detected 12 faults in Goodenough and Gerhart's specification

1989 – Schach found a fault in Meyer's specification

Goodenough and Gerhart's specification

- **constructed with great care**
- **constructed to correct Naur's specification**
- **went through two versions, carefully refereed**
- **written by experts in specifications**
- **took as much time as they needed**
- **for a product about 30 lines long**
- **still had faults!**

Writing a fault-free specification for a real product would be even more challenging…

Seven Myths of Formal Methods
(Hall 1990)

Formal methods can guarantee that software is perfect.

Formal methods are all about program proving.

Formal methods are only useful for safety-critical systems.

Formal methods require highly trained mathematicians.

Formal methods increase the cost of development.

Formal methods are unacceptable to users.

Formal methods are not used on real, large-scale software.

Can We Trust a Theorem Prover?

```
void theoremProver ( )
{
    print "This product is correct";
}
```

Difficulties with Correctness Proving

How do we find input–output specifications, loop invariants?

- **coming up with a precise statement of the specification is hard**

What if the specifications are wrong?

- **missing requirements are hard to identify**
- **ambiguous requirements are hard to clarify**

We can never be totally sure that a specification or a verification system is "correct" itself.

Examples of Formal Methods

Z

- **formal specification language designed at the University of Oxford**

VDM (Vienna Development Method)

- **developed by IBM Vienna**
- **includes specification language VDM-SL**

B-Method, Event-B

- **lower-level than Z, more focused on refinement to code than just formal specification**

…

Architecture Description Languages

Software ADLs

- **Acme (developed by CMU)**
- **AADL (standardized by SAE)**
- **C2 (developed by UCI)**
- **Darwin (developed by Imperial College London)**
- **Wright (developed by CMU)**

ISO/IEC 42010, "Systems and software engineering – Architecture description" specifies minimum requirements for ADLs

Enterprise ADLs

- **ArchiMate (now an Open Group standard),**
- **DEMO**
- **ABACUS (developed by the University of Technology, Sydney)**

Acme ADL

Entry on Wikipedia removed because could not find third party sources beyond papers and reports from the university (CMU)

CMU has seven references mentioning Acme

- **none specifically on Acme**

AADL ADL

Architecture Analysis & Design Language (AADL) standardized by SAE

- **originally Avionics Architecture Description Language**
- **derived from MetaH, an architecture description language from the Advanced Technology Center of Honeywell**

Used to model the software and hardware architecture of an embedded, real-time system

- **contains constructs for modeling both software and hardware components**
- **can be used for design documentation, for analyses (such as schedulability and flow control) or for code generation**

Use of Formal Methods in Industry

Transportation

- **Meteor line in Parisian Underground**

Space

- **Polyspace static analyzers in European Space Agency**

Aerospace

- **Airbus uses formal verification to check various aspects of software products**

Nuclear power

- **Sizewell-B nuclear power plant in the U.K.**

http://www.fm4industry.org/index.php/ExFac-HM-2
Bowen and Hinchey, Application of Formal Methods, 1995.

Formal Methods in Digital Systems

Driven by 1994 recall following the bug in the floating point division instruction of the Intel Pentium processor

AMD proof of its floating point division algorithm using the ACL2 theorem prover

Intel's Integrated Design and Verification environment, building on the Forte formal verification system

Microsoft set of formal verification tools, targeting specific issues such as driver verification

Praxis High Integrity Systems

P.E. Ross, "The Exterminators", IEEE Spectrum, September 2005.

English → Z → Spark

50% higher cost than standard rates

For electronic purse project (Mondex), 100 KSLOC project

- **warranty fix of defects for first year**
- **defect rate was 0.04/KSLOC (4 defects found)**

CICS and Z

IBM's Customer Information Control System (CICS) used Z

- **2.5 times fewer customer-reported errors;**
- **9% saving in the total development costs of the release**

Claims of quantified improvements challenged by Finney and Fenton (1996)

CICS/ESA version 3.1 (June 1990) included

- **500,000 lines of unchanged code**
- **268,000 lines of new and modified code**
- **37,000 lines were produced from Z specifications and designs**
- **11,000 lines were partially specified in Z**
- **2000 pages of formal specifications in total**

DO-178C

DO-178 is a standard for commercial aircraft software

DO-178C (2011) allows formal verification to replace certain forms of testing

- **Formal methods might be used in a very selective manner to partially address a small set of objectives, or might be the primary source of evidence for the satisfaction of many of the objectives concerned with development and verification.**

Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate, "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience," IEEE Software, May/June 2013.

DO-333

DO-178C supplement on formal methods

In DO-333, a formal method is defined as "a formal model combined with a formal analysis."

A model is formal if it has unambiguous, mathematically defined syntax and semantics.

Formal analysis techniques

- **deductive methods (such as theorem proving)**
- **model checking**
- **abstract interpretation**

Summary – Things to Remember

Myths of formal methods

Formal methods used in

- **architecture (AADL)**
- **large-scale systems (CICS)**
- **non-life-critical systems (CICS)**

FAA allowing formal methods to be used in regulated life critical systems (DO-178C, DO-333)

Questions and Answers