



L3 Mention Informatique Parcours Informatique et MIAGE

## Génie Logiciel Avancé -Advanced Software Engineering

**Deductive Verification I** 

Burkhart Wolff wolff@lri.fr

## Recall: Validation and Verification

- Validation :
  - Does the system meet the clients requirements ?
  - Will the performance be sufficient ?
  - Will the usability be sufficient ?

Do we build the right system ?

Verification: Does the system meet the specification ?

Do we build the system right ? Is it « correct » ?

#### Recall: What are the limits of tests

#### Assumptions on "Testability"

(system under test must behave deterministically, or have controlled non-determinism, must be initializable)

 Assumptions like Test-Hypothesis
 (Uniform / Regular behaviour is sometimes a "realistic" assumption, but not always)

#### Limits in perfection:

We know only up to a given "certainty" that the program meets the specification ...

## The role of formal proof

formal proofs are another technique for program verification

 based on a model of the underlying programming language, the conformance of a concrete program to its specification can be established

FOR <u>ALL</u> INPUT DATA AND <u>ALL</u> INITIAL STATES !!!

- formal proofs as verification technique can:
  - verify that a more concrete design-model "fits" to a more abstract design model (construction by formal refinement)
  - verify that a program "fits" to a concrete design model.

## Who is using formal proofs in industry?

#### Hardware Suppliers:

- INTEL: Proof of Floating Point Computation compliance to IEEE754
- INTEL: Correctness of Cash-Memory-Coherence Protocols
- > AMD: Correctness of Floating-Point-Units againt Design-Spec
- GemPlus: Verification of Smart-Card-Applications in Security

#### Software Suppliers:

- MicroSoft: Many Drivers running in "Kernel Mode" were verified
- MicroSoft: Verification of the Hyper-V OS
   (60000 Lines of Concurrent, Low-Level C Code ...)

≻ ...

## Who is using formal proofs in industry?

- For the highest certification levels along the lines of the Common Criteria, formal proofs are
  - recommended (EAL6)
  - mandatory (EAL7)

There had been now several industrial cases of EAL7 certifications ...

- For lower levels of certifications, still, formal specifications were required.
- Recently, Microsoft has agreed in a Monopoly-Lawsuit against the European Commission to provide a formal Spec of the Windows-Server-Protocols
  - the tools validating them use internally automated proofs

#### Pre-Rerquisites of Formal Proof Techniques

- A Formal Specification (MOAL, HOL, but also Z, VDM, CSP, B, ...)
  - know-how over the application domain
  - informal and formal requirements of the system
- Either a formal model of the programming language or a trusted code-generator from concrete design specs
- Tool Chains to generate, simplify, and solve large formulas (decision procedures)
- Proof Tools and Proof Checker: proofs can also be false ...

#### Nous, on le fera à la main ...

#### In the sequel, we concentrate on

## **Deductive Verification**

## (Proof Techniques)

B. Wolff - GLA - Deductive Verification

#### The specification in UML/MOAL (Classes in USE Notation):

**class** Triangles inherits\_from Shapes

#### attributes

- a : Integer
- b : Integer
- c : Integer

#### operations

```
mk(Integer,Integer,Integer):Triangle
is_Triangle(): triangle
end
```

#### Standard example : Triangle

#### The specification in UML/OCL (Classes in USE Notation):

#### Standard example: Triangle

```
procedure triangle(j,k,l : positive) is
eq: natural := 0;
begin
if j + k \le 1 or k + 1 \le j or 1 + j \le k then
   put("impossible");
else if j = k then eq := eq + 1; end if;
     if j = 1 then eq := eq + 1; end if;
     if l = k then eq := eq + 1; end if;
     if eq = 0 then put("quelconque");
    elsif eq = 1 then put("isocele");
    else put ("equilateral");
    end if;
end if;
end triangle;
```

## Program Example : Exponentiation

```
Program_1:
    (* pre : N≥0 *)
    S:=1; P:=N;
    while P >= 1 loop S:= S*X; P:= P-1; end loop;
    (* post: S = X<sup>N</sup> *)
Program_2:
    (* pre : N≥0 *)
    S:=1; P:= N;
    while P >= 1 loop
        if P mod 2 <> 0 then P := P-1; S := S*X; end if;
        S:= S*S; P := P div 2;
    end loop;
    (* post: S = X<sup>N</sup> *)
```

These programs have the following characteristics:

- one is more efficient, but more complex
- But both have the same specification !

# How to PROVE that programs meet the specification ?

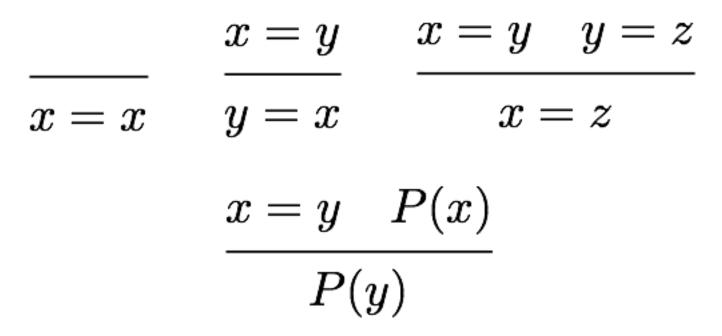
B. Wolff - GLA - Deductive Verification

 An Inference System (or Logical Calculus) allows to infer formulas from a set of elementary facts (axioms) and inferred facts by rules:

$$\frac{A_1 \quad \dots \quad A_n}{A_{n+1}}$$

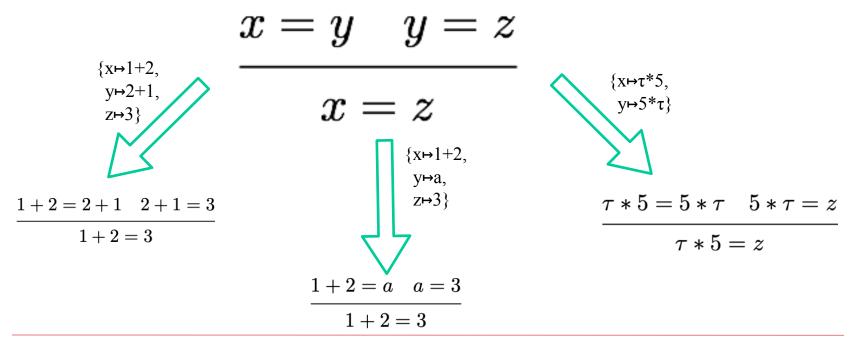
- "from the assumptions  $A_1$  to  $A_n$ , you can infer the conclusion  $A_{n+1}$ ." A rule with n=0 is an elementary fact. Variables occurring in the formulas  $A_n$  can be arbitrarily substituted.
- Assumptions and conclusions are terms in a logic containing variables

 An Inference System for the equality operator (or "Equational Logic") looks like this:



• where the first rule "reflexivity" is an elementary fact.

The variables in an inference rule can be replaced by a substitution. The substituted inference rule is called an instance (of this rule).



B. Wolff - GLA - Deductive Verification

A Formal Proof (or : Derivation)
 is a tree with rule instances as nodes

f(a,b) = a	$\underbrace{f(a,b) = a  f(f(a,b),b) = c}_{}$	
a = f(a, b)	f(a,b)=c	
	a = c	$\overline{g(a) = g(a)}$
	g(a) = g(c)	

• The non-elementary facts at the leaves are the global assumptions (here f(a,b) = a and f(f(a,b),b) = c).

As a short-cut, we also write for a derivation:

$$\{A_1,\ldots,A_n\}\vdash A_{n+1}$$

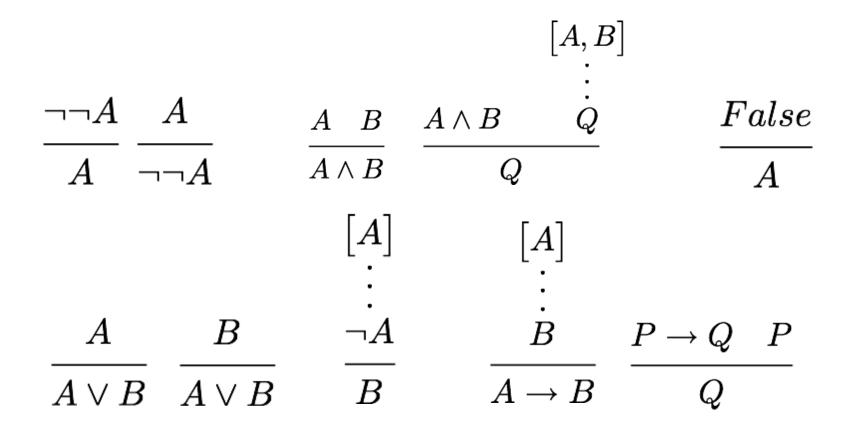
• ... or generally speaking: from global assumptions A to a theorem (in theory E)  $\varphi$ :

$$\Gamma \vdash_E \phi$$

This is what theorems are: derivable facts from assumptions in a certain logical system ...

## A Proof System for Propositional Logic

PL + E + Arithmetics (A) in so-called natural deduction:



B. Wolff - GLA - Deductive Verification

Now, can we build a

#### Logic for Programs ???

Now, can we build a

Logic for Programs ???

Well, yes ! There are actually lots of possibilities ...

We consider the Hoare-Logic (Sir Anthony Hoare ...), technically an inference system PL + E + A + Hoare

- Basis: The mini-language "IMP", (following Glenn Wynskell's Book)
- We have the following commands (cmd)
  - the empty command SKIP
  - > the assignment x := E  $(x \in V)$
  - the sequential compos.
- $C_1; C_2$
- > the conditional IF cond THEN  $c_1$  ELSE  $c_2$
- the loop WHILE cond DO c

where c,  $c_1$ ,  $c_2$ , are cmd's, V variables,

E an arithmetic expression, and cond a boolean expression.

- Core Concept: A Hoare Triple consisting ...
  - ➤ of a pre-condition
  - $\succ$  a post-condition Q
  - and a piece of program cmd
  - the triple (P,cmd,Q) is written:

$$\vdash \{P\} \ cmd \ \{Q\}$$

P and Q are formulas over the variables V, so they can be seen as set of possible states. Hoare Logic vs. Symbolic Execution

• Hoare Logic is also based notion of a *symbolic state*.

state<sub>sym</sub> = 
$$V \rightarrow Set(D)$$

#### Hoare Logic vs. Symbolic Execution

• Intuitively:

 $\vdash \{P\} \ cmd \ \{Q\}$  -

#### means:

If a program cmd starts in a state admitted by P if it terminates, that the program must reach a state that satisfies Q.

PL + E + A + Hoare (simplified binding) at a glance:

 $\vdash \{P \land cond\} \ c \ \{Q\} \quad \vdash \{P \land \neg cond\} \ d \ \{Q\}$ 

 $\vdash \{P\}$  IF cond THEN c ELSE  $d\{Q\}$ 

 $\frac{\vdash \{P\} \ c \ \{Q\} \ \vdash \{Q\} \ d \ \{R\}}{\vdash \{P\} \ c; \ d \ \{R\}} \quad \frac{\vdash \{P \land cond\} \ c \ \{P\}}{\vdash \{P\} \ \text{WHILE} \ cond \ \text{DO} \ c \ \{P \land \neg cond\}}$ 

 $\frac{P \to P' \quad \vdash \{P'\} \ cmd \ \{Q'\} \quad Q' \to Q}{\vdash \{P\} \ cmd \ \{Q\}}$ 

#### Verification : Test or Proof

#### Test

 Requires Testability of Programs (initialisable, reproducible behaviour, sufficient control over non-determinism)

Can be also Work-Intensive !!!

- Requires Test-Tools
- Requires a Formal Specification
- Makes Test-Hypothesis, which can be hard to justify !

#### Summary

## Formal Proof

- Can be very hard up to infeasible (no one will probably ever prove correctness of MS Word!)
- Proof Work typically exceeds programming work by a factor 10!
- Tools and Tool-Chains necessary
- Makes assumptions on language, method, tool-correctness, too !

## Validation : Test or Proof (end)

#### Test and Proof are Complementary ...

- analysis to formal proof of "deep system properties"
- In practice, a good "verification plans" will be necessary to get the best results with a (usually limited) budget !!!
  - detect parts which are easy to test
  - detect parts which are easy to prove
  - good start: maintained formal specification

#### Hoare - Logic: Outlook

Can we be sure, that the logical systems are consistent ?

Well, yes, practically. (See Hales Article in AMS: "Formal Proof", 2008. http://www.ams.org/ams/press/hales-nots-dec08.html)

Can we ever be sure, that a specification "means" what we intend ?

Well, no. But when can we ever be entirely sure that we know what we have in mind? But at least, we can gain confidence validating specs, i.e. by animation and test, thus, by experimenting with them ...

#### $\vdash \{P \land \neg cond\} \text{ WHILE } cond \text{ DO } c \ \{P \land \neg cond\}$

## $P = P' \quad \vdash \{P'\} \ cmd \ \{Q'\} \quad Q' = Q$ $\vdash \{P\} \ cmd \ \{Q\}$