

## Mechanical Verification

Dr Malcolm Newey  
Department of Computer Science  
The Australian National University

E-mail: Jim.Grundy@anu.edu.au

## What This Unit Will Cover

- Formal Methods and Their Applications
- Higher-Order Logic
- The HOL Theorem Prover
- Embedding Logics and Languages in HOL
- Hardware Verification in HOL
- Protocol Verification in HOL

Mechanical Verification: Lecture 1 2000 1

Mechanical Verification: Lecture 1 2000 2

## What are Formal Methods

*Formal Methods* is the application of logic to the development of 'correct' computer systems.

*Correctness* is classically viewed as two separate problems, *verification* and *validation*.

**Verification:** Are we building the product right?

Can we use logic to help us ensure that the product built faithfully implements its specification?

**Validation:** Are we building the right product?

Can we use logic to help us ensure that the specification is complete, consistent, and accurately captures the customer's requirements?

Mechanical Verification: Lecture 1 2000 3

## Why Use Formal Methods?

Formal methods are typically employed for the following reasons:

- they are required by the contract or by law
- they are expected to save development costs
- the product cannot be made sufficiently reliable otherwise

## Where are Formal Methods Used?

- Microprocessor Design
- Cache Coherency Protocols
- Telecommunications Protocols
- Rail and Track Signalling
- Security Protocols
- Automotive Companies

Mechanical Verification: Lecture 1 2000 4

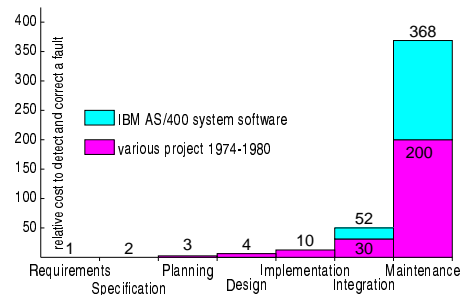
## Formal Specification

Formal specification is an example of formal methods applied to *validation*.

- The language of logic provides an unambiguous method of recording the specification.
- The process of writing a formal specification helps uncover any ambiguity and incompleteness.
- We can reason about a formal specification to check that the system specified will possess other desired properties.

Mechanical Verification: Lecture 1 2000 5

## Formal Specification — The Financial Case For



It is 100 times cheaper to fix a specification fault when it is made!

Mechanical Verification: Lecture 1 2000 6

## Formal Specification Techniques

Formal specification techniques can be partitioned into two broad classes:

**Algebraic Specification:** The object to be described is considered as a collection of actions. Equations are then used to show the desired inter-relations between those actions.

Well known examples include OBJ.

**Model Based Specification:** The object to be described is modelled in some expressive logic. This is like creating an extremely high-level, nonexecutable implementation.

Well known examples include Z and VDM.

Mechanical Verification: Lecture 1 2000 7

## Algebraic Specification Example: A Stack

**Signature:** The signature lists the actions of the object and their types.

empty:  $\alpha$  STACK  
top:  $\alpha$  STACK  $\rightarrow$   $\alpha$   
pop:  $\alpha$  STACK  $\rightarrow$   $\alpha$  STACK  
push:  $\alpha \times \alpha$  STACK  $\rightarrow$   $\alpha$  STACK  
depth:  $\alpha$  STACK  $\rightarrow$   $\mathbb{N}$

**Equations:** The equations define the inter-relationships between the actions.

$\text{depth}(\text{empty}) \stackrel{\text{def}}{=} 0$   
 $\text{depth}(\text{push}(e,s)) \stackrel{\text{def}}{=} \text{depth}(s) + 1$   
 $\text{top}(\text{push}(e,s)) \stackrel{\text{def}}{=} e$   
 $\text{pop}(\text{push}(e,s)) \stackrel{\text{def}}{=} s$

Mechanical Verification: Lecture 1 2000 8

### Model Based Specification Example: A Stack

Stacks of  $\alpha$ s are modelled by functions from the naturals to  $\alpha$ s such that if there are  $n$  things on a stack then the domain of the function is  $\{0..n-1\}$ .

$$\alpha\text{STACK} \stackrel{\text{def}}{=} \{f \in \mathbb{N} \rightarrow \alpha \mid \text{dom}(f) = \{n \in \mathbb{N} \mid n < \#(f)\}\}$$

Each function will map 0 to the bottom element of the stack it models, 1 to the second bottom element, and so on.



Example: The stack 

Kyle
Kenny
Kartman

 is represented by the function

0  $\mapsto$  Kartman  
1  $\mapsto$  Kenny  
2  $\mapsto$  Kyle

which is itself the set  $\{(0, \text{Kartman}), (1, \text{Kenny}), (2, \text{Kyle})\}$ .

### Models of Stack Operations

An empty stack is represented by the empty function, and the depth of a stack is the size of its representation.

$$\text{empty} \stackrel{\text{def}}{=} \{\}$$

$$\text{depth}(s) \stackrel{\text{def}}{=} \#(s)$$

Pushing an element adds another mapping to the function.

$$\text{push}(e, s) \stackrel{\text{def}}{=} s \cup \{(\#(s), e)\}$$

The top element of a stack (function)  $s$  of depth  $n$  is  $s(n-1)$ .

$$(\text{depth}(s) \neq 0) \sqsupset (\text{top}(s) \stackrel{\text{def}}{=} s(\#(s) - 1))$$

To pop a stack of depth  $n$ , remove the  $n-1$  mapping from its representation.

$$(\text{depth}(s) \neq 0) \sqsupset (\text{pop}(s) \stackrel{\text{def}}{=} \{(n, e) \in s \mid n < \#(s) - 1\})$$

### Proof Obligations About The Model

The functions should preserve the stack structure defined:

- $\text{empty} \in \alpha\text{STACK}$
- $\text{push}(e, s) \in \alpha\text{STACK}$
- $(\text{depth}(s) \neq 0) \sqsupset (\text{pop}(s) \in \alpha\text{STACK})$

The defining properties of the algebraic specification should also hold:

- $\text{depth}(\text{empty}) = 0$
- $\text{depth}(\text{push}(e, s)) = \text{depth}(s) + 1$
- $\text{top}(\text{push}(e, s)) = e$
- $\text{pop}(\text{push}(e, s)) = s$

### A Comparison of Specification Styles

Algebraic Specifications

- ✓ Simple to read.
- ✓ Avoids construction of a model.
- ✗ Limited expressiveness of language.

Model Based Specifications

- ✓ More expressive language.
- ✗ Can be harder to read.
- ✗ Can lead to 'over-specification' of the problem.

### Formal Verification

Formal verification is an example of formal methods applied to *verification*.

- Formal verification is a process of constructing a proof that a computer system will behave in accordance with its specification.

The 'standard' verification technique is testing, but ...

Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.

Edsgar W. Dijkstra

Suppose you want to test a 64-bit floating point division routine. There are  $2^{128}$  combinations. At 1 test/ $\mu$ s, it'll take  $10^{25}$  years!

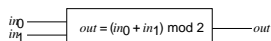
### The Formal Verification Process

- Describe the property you want to hold (the formal specification).
- Describe the implementation you hope has it.
- Construct a proof to this effect.

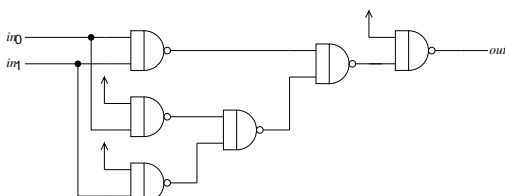
### Formal Verification Requires

- A language for describing both specifications and implementations.
- A deductive calculus for proving propositions in this language.

### Specification of a Half-Adder



### Is This Implementation Correct?



### Limitations of Formal Verification

Just because you have proved something correct doesn't mean it will work!

There are gaps where formal verification connects with the real world.

- Does the specification actually capture the designer's intentions?  
Specifications must be simple and abstract.  
The following is *not* a good specification for a half-adder:  
$$\text{out} = (in_0 \vee in_1) \wedge \neg(in_0 \wedge in_1)$$
- Does the implementation in the real world behave like the model?
  - Can  $in_1$  drive three input lines?
  - What happens if the wires are fabricated too close together?  
(Do we need to model quantum effects on the silicon surface?)