## NICTA

#### COMP 4161 NICTA Advanced Course

#### Advanced Topics in Software Verification

Simon Winwood, Toby Murray, June Andronick, Gerwin Klein



#### Slide 1

Content

- → Intro & motivation, getting started with Isabelle
- → Foundations & Principles
  - Lambda Calculus
  - Higher Order Logic, natural deduction
  - Term rewriting
- → Proof & Specification Techniques
  - Inductively defined sets, rule induction
  - Datatypes, recursion, induction
  - More recursion, Calculational reasoning

#### • Hoare logic, proofs about programs

Locales, Presentation



### Last Time



- → Code generation
- → Syntax of a simple imperative language
- → Operational semantics
- → Program proof on operational semantics

Slide 3

#### Proofs about Programs



#### Now we know:

- → What programs are: Syntax
- → On what they work: State
- → How they work: Semantics

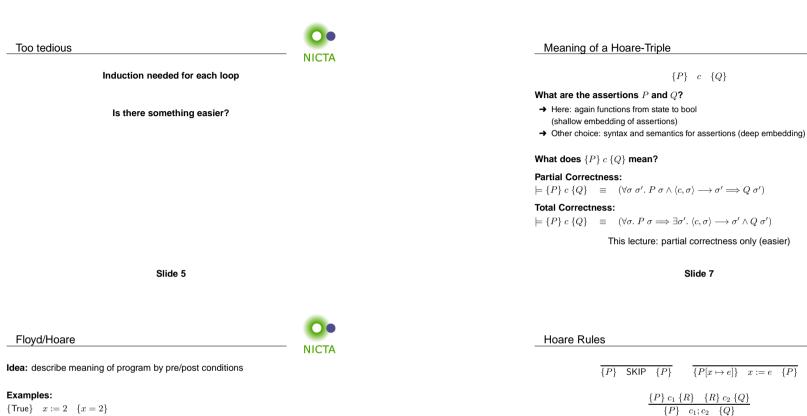
So we can prove properties about programs

#### Example:

Show that example program from last lecture implements the factorial.

**lemma** (factorial,  $\sigma$ )  $\longrightarrow \sigma' \Longrightarrow \sigma' B = fac (\sigma A)$ (where fac 0 = 0, fac (Suc n) = (Suc n) \* fac n)

Slide 2



 $\begin{array}{l} \{ \mathsf{True} \} \quad x := 2 \quad \{ x = 2 \} \\ \{ y = 2 \} \quad x := 21 * y \quad \{ x = 42 \} \end{array}$ 

 $\{x=n\} \quad \mathsf{IF} \; y < 0 \; \mathsf{THEN} \; x := x+y \; \mathsf{ELSE} \; x := x-y \quad \{x=n-|y|\}$ 

 $\{A = n\}$  factorial  $\{B = fac n\}$ 

Proofs: have rules that directly work on such triples



3

 $\frac{\{P \land b\} c_1 \{Q\} \quad \{P \land \neg b\} c_2 \{Q\}}{\{P\} \quad \mathsf{IF} \ b \ \mathsf{THEN} \ c_1 \ \mathsf{ELSE} \ c_2 \quad \{Q\}}$ 

 $\frac{\{P \land b\} c \{P\} \quad P \land \neg b \Longrightarrow Q}{\{P\} \quad \text{WHILE } b \text{ DO } c \text{ OD } \{Q\}}$ 

 $\frac{P \Longrightarrow P' \quad \{P'\} c \{Q'\} \quad Q' \Longrightarrow Q}{\{P\} \quad c \quad \{Q\}}$ 

NICTA

NICTA

Slide 8

4



#### Hoare rule application seems boring & mechanical.

Automation?

**Problem:** While – need creativity to find right (invariant) P

#### Solution:

→ annotate program with invariants

Nicer, but still kind of tedious

→ then, Hoare rules can be applied automatically

#### Example:

 $\{M = 0 \land N = 0\}$  WHILE  $M \neq a \text{ INV } \{N = M * b\} \text{ DO } N := N + b; M := M + 1 \text{ OD } \{N = a * b\}$ 

Slide 11

#### Weakest Preconditions



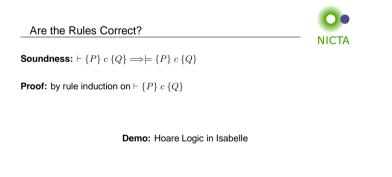
pre c Q = weakest P such that  $\{P\} c \{Q\}$ 

#### With annotated invariants, easy to get:

pre SKIP $Q$	=	Q
pre $(x := a) Q$	=	$\lambda \sigma. \; Q(\sigma(x:=a\sigma))$
pre $(c_1; c_2) Q$	=	pre $c_1$ (pre $c_2 Q$ )
pre (IF $b$ THEN $c_1$ ELSE $c_2$ ) $Q$	=	$\lambda \sigma. \ (b \longrightarrow \operatorname{pre} c_1 \ Q \ \sigma) \land$
		$(\neg b \longrightarrow pre \ c_2 \ Q \ \sigma)$
pre (WHILE $b$ INV $I$ DO $c$ OD) $Q$	=	Ι

# 

Slide 9



Slide 10

#### Verification Conditions

NICTA

 $\{ pre \ c \ Q \} \ c \ \{Q\}$  only true under certain conditions

These are called verification	n conditions vc $c Q$ :
-------------------------------	-------------------------

vc SKIP $Q$	=	True
$\mathrm{vc}\;(x:=a)\;Q$	=	True
$vc\;(c_1;c_2)\;Q$	=	$vc \ c_2 \ Q \land (vc \ c_1 \ (pre \ c_2 \ Q))$
vc (IF $b$ THEN $c_1$ ELSE $c_2$ ) $Q$	=	$vc \ c_1 \ Q \land vc \ c_2 \ Q$
vc (WHILE $b \; {\rm INV} \; I \; {\rm DO} \; c \; {\rm OD}) \; Q$	=	$(\forall \sigma. \ I\sigma \wedge b\sigma \longrightarrow pre \ c \ I \ \sigma) \wedge$
		$(\forall \sigma. \ I\sigma \land \neg b\sigma \longrightarrow Q \ \sigma) \land$
		$vc \; c \; I$

 $\mathsf{vc}\; c\; Q \wedge (\mathsf{pre}\; c\; Q \Longrightarrow P) \Longrightarrow \{P\}\; c\; \{Q\}$ 

#### Slide 13

	0.
Syntax Tricks	
	NICIA

→  $x := \lambda \sigma$ . 1 instead of x := 1 sucks

→  $\{\lambda\sigma. \sigma x = n\}$  instead of  $\{x = n\}$  sucks as well

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

#### Choices:

- → declare program variables with each Hoare triple
  - nice, usual syntax
  - works well if you state full program and only use vcg
- → separate program variables from Hoare triple (use extensible records), indicate usage as function syntactically
  - more syntactic overhead
  - program pieces compose nicely

Slide 14

#### Records in Isabelle



NICTA

Records are a tuples with named components

#### Example:

		record A =	a :: nat b :: int
→ Selectors:	$a::A\Rightarrow nat,$	$b::A\Rightarrow int,$	$a \ r = Suc \ 0$
→ Constructo	rs: (  a = Suc	0, b = -1	
→ Update:	r(  a := Suc 0  )		

#### Records are extensible:

record B = A + c :: nat list

(| a = Suc 0, b = -1, c = [0, 0] |)

Slide 15

#### Arrays

#### Depending on language, model arrays as functions:

→ Array access = function application:

a[i] = a i

→ Array update = function update: a[i] :== v = a :== a(i:= v)

#### Use lists to express length:

→ Array access = nth: a[i] = a ! i

- → Array update = list update: a[i] :== v = a :== a[i:= v]
- → Array length = list length: a.length = length a

5.1.4			
Pointers			NICTA
Choice 1			
datatype	ref	= Ref int   Null	
types	heap	= int $\Rightarrow$ val	
datatype	val	= Int int   Bool bool   Struct_x int int bool	
	ccess: *	f p = the_Int (hp (the_addr p)) p :== v = hp :== hp ((the_addr p) := v)	

- → a bit klunky
- → gets even worse with structs
- → lots of value extraction (the\_Int) in spec and program

Slide 17

Pointers	0•
Fointers	NICTA
	шент

#### Choice 2 (Burstall '72, Bornat '00)

struct with next pointer and element

datatype ref = Ref int | Null

- types  $next_hp = int \Rightarrow ref$
- elem\_hp = int  $\Rightarrow$  int types
- → next :: next\_hp, elem :: elem\_hp, p :: ref
- → Pointer access: p→next = next (the\_addr p)
- → Pointer update: p→next :== v = next :== next ((the\_addr p) := v)
- → a separate heap for each struct field
- → buys you p→next  $\neq$  p→elem automatically (aliasing)
- → still assumes type safe language

NICTA

**D**емо

Slide 19

#### We have seen today ...



- → Hoare logic rules
- → Soundness of Hoare logic
- → Verification conditions
- → Example program proofs
- → Arrays, pointers

Slide 18