	0	•
N	IC.	ТΑ

NICTA

COMP 4161 NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray, Rafal Kolanski

С

Slide 1

Last Time

- ➔ Weakest preconditions
- → Verification conditions
- → Example program proofs
- → Arrays, pointers
- → Hard part: finding invariants

Content	
	NICTA
→ Intro & motivation, getting started	[1]
→ Foundations & Principles	
 Lambda Calculus, natural deduction 	[1,2]
Higher Order Logic	[3 ^a]
Term rewriting	[4]
➔ Proof & Specification Techniques	
• Isar	[5]
 Inductively defined sets, rule induction 	[6 ^b]
 Datatypes, recursion, induction 	[7 ^c , 8]
 Calculational reasoning, code generation 	[9]
Hoare logic, proofs about programs	[10 ^d ,11,12]

^aa1 due; ^ba2 due; ^csession break; ^da3 due

Slide 3

Program Verification



So far:

- → have verified functional programs written in HOL
- → generated ML/Haskell/OCaml code for them
- → learned about verifying imperative programs with Hoare Logic

Next few lectures:

→ real C programs

Slide 2

Slide 4

1



Main new problems in verifying C programs:

- → expressions with side effects
- → more control flow (do/while, for, break, continue, return)
- ➔ local variables and blocks
- ➔ functions & procedures
- → concrete C data types
- → C memory model and C pointers

C is not a nice language for reasoning.

Things are going to get ugly.

Slide 5

Approach NICTA

Approach for verifying C programs:

Translate into existing, clean imperative language in Isabelle.

Simpl:

- → generic imperative language by Norbert Schirmer, TU Munich
- → state space and basic expressions/statements can be instantiated
- → has operational semantics
- $\label{eq:Hoare}$ Hoare logic with soundness and completeness proof
- → automated vcg
- → available from the Archive of Formal Proofs http://afp.sf.net



NICTA

type_synonym 's bexp = "'s set"

Commands in Simpl

datatype ('s, 'p, 'f) com =
 Skip
 Basic "'s => 's"
 Spec "('s * 's) set"
 Seq "('s , 'p, 'f) com" "('s, 'p, 'f) com"
 Cond "'s bexp" "('s, 'p, 'f) com" "('s, 'p, 'f) com"
 While "'s bexp" "('s, 'p, 'f) com"
 Call 'p
 DynCom "'s => ('s, 'p, 'f) com"
 Guard 'f "'s bexp" "('s, 'p, 'f) com"
 Throw
 Catch "('s, 'p, 'f) com" "('s, 'p, 'f) com"

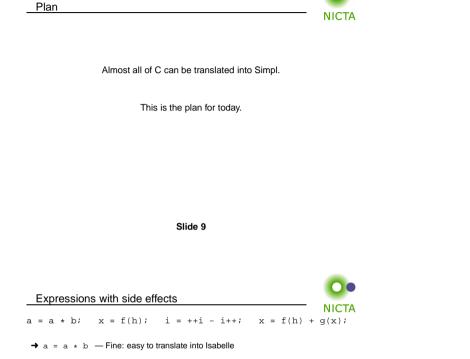
's = state, 'p = procedure names, 'f = faults

Slide 7



DEMO: SIMPL

Slide 8



- \Rightarrow x = f(h) Fine: may have side effects, but can be translated sanely.
- → i = ++i i++ Seriously? What does that even mean? Make this an error, force programmer to write instead: i0 = i; i++; i = i - i0; (or just i = 1)
- → x = f(h) + g(x) Ok if g and h do not have any side effects ⇒ Prove all functions in expressions are side-effect free

Alternative: explicitly model nondeterministic order of execution in expressions.

Slide 10

Control flow



do { c } while (condition);

Already can treat normal while-loops! Automatically translate into:

c; while (condition) { c }

Similarly:

for (init; condition; increment) { c }

becomes

init; while (condition) { c; increment; }

Slide 11

More control flow: break/continue while (condition) { foo; if (Q) continue; bar; if (P) break; } Non-local control flow: continue goes to condition, break goes to end. Can be modelled with exceptions: → throw exception continue, catch at end of body.

→ throw exception break, catch after loop.

Exceptions	
Exceptions	
	NICTA
Do not exist in C, but can be used to model C constructs.	

Exceptions can be modelled with two kinds kinds of state:

- → normal states as before
- → abrupt states an exception was raised, normal commands are skipped.

Simpl commands:

- → throw: switch to abrupt state
- → try { c1 } catch { c2 }: if c1 terminates abruptly, execute c2, otherwise execute only c1.

Use state to store which exception was thrown.

Slide 13

```
Break/continue
Dreak/continue example becomes:

try {
    while (condition) {
        try {
            foo;
            if (Q) { exception = 'continue'; throw; }
            bar;
            if (P) { exception = 'break'; throw; }
        } catch { if (exception == 'continue') SKIP else throw; }
    }
} catch { if (exception == 'break') SKIP else throw; }
```

This is not C any more. But it models C behaviour!

Need to be careful that only the translation has access to exception state.

Slide 14

Return



NICTA

if (P) return x;
foo;
return y;

Similar non-local control flow. Similar solution: use throw/try/catch

```
try {
    if (P) { return_val = x; exception = 'return'; throw; }
    foo;
    return_val = y; exception = 'return'; throw;
} catch {
    SKIP
}
```

Slide 15

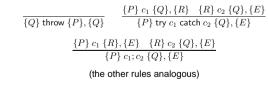
Hoare Rules for Exceptions

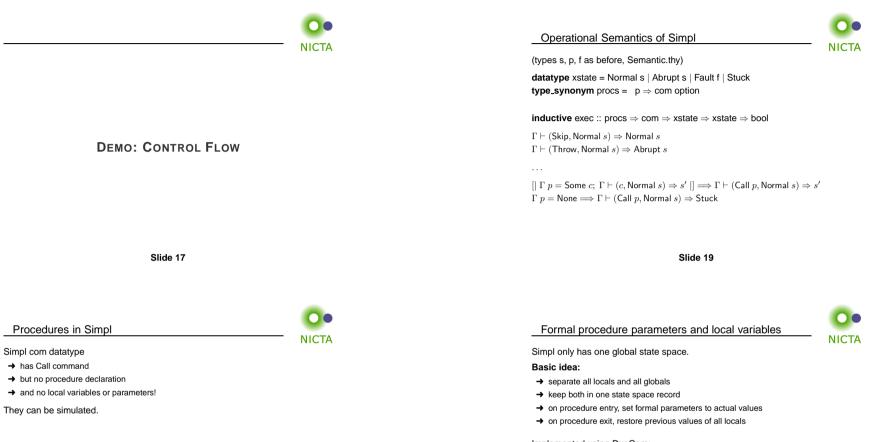


 $\{P\} f \{Q\}, \{E\}$

- If \boldsymbol{P} holds initially, and
- \rightarrow f terminates in state Normal s, then Q s;
- \rightarrow f terminates in state Abrupt s, then E s

Hoare Rules:





Slide 18

Implemented using DynCom:

call init body restore result =

DynCom (λ s. init; body; DynCom (λ t. restore s t; result t))

Example: for procedure $f(x) = \{ r = x + 2 \}$

 $y = \text{CALL } f(7) \quad \equiv \text{call } (x = 7) \ (r = x + 2) \ (\lambda s \ t. \ s \ (| \ \text{globals} := \text{globals} \ t \ |)) \ (\lambda t. \ y = r \ t)$

Slide 20

Copyright NICTA 2012, provided under Creative Commons Attribution License

Verifying Procedures NICTA

Simple idea: replace/inline body. Does not work for recursion.

Instead:

- → introduce assumed specifications for procedures
- → outside call: no specification known, user provided
- → but: can assume current specification for recursive call
- → works like induction
- → is proved by induction on the recursive call depth

We have seen today ..



- → C control flow
- → Exceptions with Hoare logic rules
- → C functions and procedures with Hoare logic rules

Slide 21 Slide 23

DEMO: PROCEDURES