



COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray, Rafal Kolanski

C

Slide 1



Last Time

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

Slide 2



Content

- 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 4

C



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



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
- Hoare logic with soundness and completeness proof
- automated vcg
- available from the Archive of Formal Proofs <http://afp.sf.net>

Slide 6

Commands in Simpl



```
type_synonym 's bexp = "'s set"
```

```
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

Plan



Almost all of C can be translated into Simpl.

This is the plan for today.

Slide 9

Expressions with side effects



```
a = a * b;   x = f(h);   i = ++i - i++;   x = f(h) + g(x);
```

→ `a = a * b` — Fine: easy to translate into Isabelle

→ `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.

Slide 12

Exceptions



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

Exceptions can be modelled with two 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



Break/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



```
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



Need new kind of Hoare triples to model normal and abrupt state:

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

If P holds initially, and

- f terminates in state Normal s , then Q s ;
- f terminates in state Abrupt s , then E s

Hoare Rules:

$$\frac{\{P\} c_1 \{Q\}, \{R\} \quad \{R\} c_2 \{Q\}, \{E\}}{\{Q\} \text{ throw } \{P\}, \{Q\}} \quad \frac{\{P\} c_1 \{Q\}, \{R\} \quad \{R\} c_2 \{Q\}, \{E\}}{\{P\} \text{ try } c_1 \text{ catch } c_2 \{Q\}, \{E\}}$$

$$\frac{\{P\} c_1 \{R\}, \{E\} \quad \{R\} c_2 \{Q\}, \{E\}}{\{P\} c_1; c_2 \{Q\}, \{E\}}$$

(the other rules analogous)

Slide 16



DEMO: CONTROL FLOW

Slide 17



Procedures in Simpl

Simpl com datatype

- has Call command
- but no procedure declaration
- and no local variables or parameters!

They can be simulated.

Slide 18

Operational Semantics of Simpl



(types s, p, f as before, Semantic.thy)

datatype $xstate = Normal\ s \mid Abrupt\ s \mid Fault\ f \mid Stuck$

type_synonym $procs = p \Rightarrow com\ option$

inductive $exec :: procs \Rightarrow com \Rightarrow xstate \Rightarrow xstate \Rightarrow bool$

$\Gamma \vdash (Skip, Normal\ s) \Rightarrow Normal\ s$

$\Gamma \vdash (Throw, Normal\ s) \Rightarrow Abrupt\ s$

...

$[\![\Gamma\ p = Some\ c; \Gamma \vdash (c, Normal\ s) \Rightarrow s']\!] \Longrightarrow \Gamma \vdash (Call\ p, Normal\ s) \Rightarrow s'$

$\Gamma\ p = None \Longrightarrow \Gamma \vdash (Call\ p, Normal\ s) \Rightarrow Stuck$

Slide 19



Formal procedure parameters and local variables

Simpl only has one global state space.

Basic idea:

- separate all locals and all globals
- keep both in one state space record
- on procedure entry, set formal parameters to actual values
- on procedure exit, restore previous values of all locals

Implemented using DynCom:

call $init\ body\ restore\ result =$

$DynCom\ (\lambda s. init; body; DynCom\ (\lambda t. restore\ s\ t; result\ t))$

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

$y = CALL\ f(7) \equiv call\ (x = 7)\ (r = x + 2)\ (\lambda s\ t. s\ (\mid\ globals := globals\ t\ \mid))\ (\lambda t. y = r\ t)$

Slide 20

Verifying Procedures



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

Slide 21



DEMO: PROCEDURES

Slide 22

We have seen today ...



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

Slide 23