COMP4161
Advanced Topics in Software Verification

INV & Exam Prep

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

COMP4161
Advanced Topics in Software Verification

INV

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

Practice with invariants!

Recall:

- ➜ invariants are needed to automate the application of hoare rules
- ➜ they are used by the weakest precondition calculus to deal with loops

Recall:

- ➜ an invariant needs to be “enough” (to prove the postcondition)
- ➜ an invariant needs to be an invariant
  - ➜ “true before the loop”
  - ➜ “if true at the start of an iteration, still true after one iteration”

Weakest precondition - recall

$(P \Longrightarrow pre\ (i_0;\ i_1;\ i_2;)\ Q) \Longrightarrow \{\ P\ \}\ i_0;\ i_1;\ i_2;\ \{\ Q\ \}$

$\{\ P\ \}$

$pre\ i_0\ (pre\ i_1\ (pre\ i_2\ Q)) = pre\ i_1;\ i_2;\ i_3;\ Q$

$i_0;$

$pre\ i_1\ (pre\ i_2\ Q)$

$i_1;$

$pre\ i_2\ Q$

$i_2;$

$\{\ Q\ \}$

Invariant - recall

$\{ P \}$

$P \implies I$ ("true before the loop")

?? $pre\ (WHILE\ b\ INV\ I\ DO\ c\ OD) = I$

$WHILE\ b\ INV\ I$

$I \land b \implies pre\ c\ I$

("if true at the start of an iteration,")

("still true after one iteration")

DO

c

OD

$I \land \neg b \implies Q$ ("enough")

$\{ Q \}$

Example 1

$\{ a \geq 0 \ \wedge \ b \geq 0 \}$
$A := 0;$
$B := 0;$
INV { B = b * A}
WHILE $A \neq a$
DO
  $B := B + b;$
  $A := A + 1$
OD
$\{ B = b * a \}$

$A =$	0	1	2	3	4	...
$B =$	0	b	b+b	b+b+b	b+b+b+b	...

Example 1

$\{ a \geq 0 \ \wedge \ b \geq 0 \}$
$A := 0;$
$B := 0;$ $\qquad 0 = b * 0 \quad \checkmark$
INV { B = b * A}
WHILE $A \neq a$ $\qquad B = b * A \wedge A \neq a \longrightarrow B + b = b * (A + 1)$
DO $\qquad = b * A + b$
$\quad B := B + b;$ $\qquad = B + b \quad \checkmark$
$\quad A := A + 1$
OD $\qquad B = b * A \wedge A = a \longrightarrow B = b * a \quad \checkmark$
$\{ B = b * a \}$

Example 2

$\{\ a \geq 0 \ \wedge\ b \geq 0\ \}$
$A := 0;$
$B := 0;$ $\quad 0 = b * 0 \quad \checkmark$
INV { B = b * A}
WHILE $A < a$ $\quad B = b * A \wedge A < a \longrightarrow B + b = b * (A + 1)$
DO $\quad = b * A + b$
$\quad B := B + b;$ $\quad = B + b \quad \checkmark$
$\quad A := A + 1$
OD $\quad B = b * A \wedge A \geq a \longrightarrow B = b * a \quad ???$
$\{\ B = b * a\ \}$

Example 2

$\{ a \geq 0 \ \wedge \ b \geq 0 \}$
$A := 0;$
$B := 0;$ $\qquad 0 = b * 0 \wedge 0 \leq a \quad \checkmark$
INV $\{ B = b * A \wedge A \leq a \}$
WHILE $A < a$ $\qquad B = b * A \wedge A < a \longrightarrow B + b = b * (A + 1)$
DO $\qquad \wedge A \leq a \qquad \wedge A + 1 \leq a \quad \checkmark$
$\quad B := B + b;$
$\quad A := A + 1$
OD $\qquad B = b * A \wedge A \geq a \longrightarrow B = b * a$
$\{ B = b * a \}$ $\qquad \wedge A \leq a \quad \checkmark$

Example 3

$\{\ a \geq 0\ \wedge\ b > 0\ \}$

$A := a;$ $\quad A =$ a, a-1, a-2, a-3, ...

$B := 1;$ $\quad B =$ 1, b, b*b, b*b*b, ...

$= b^3 = b^{a-A}$

$1 = b^{a-a}$

$\text{INV}\ \{\ B = b^{a-A}\}$

$\text{WHILE}\ A \neq 0$ $\qquad B = b^{a-A} \wedge A \neq 0 \longrightarrow\ B * b = b^{a-(A-1)}$

DO

$\quad B := B * b;$

$\quad A := A - 1$

OD $\qquad B = b^{a-A} \wedge A = 0\ \longrightarrow\ B = b^a$

$\{\ B = b^a\ \}$

Example 3

$\{ a \geq 0 \ \wedge \ b > 0 \}$

$A := a;$

$B := 1;$

$A =$	a	a-1	a-2	a-3	...
$B =$	1	b	b*b	b*b*b	...
				$= b^3 = b^{a-A}$	

$1 = b^{a-a}$

INV $\{ B = b^{a-A} \wedge A \leq a \}$

WHILE $A \neq 0$

$B = b^{a-A} \wedge A \neq 0 \longrightarrow B * b = b^{a-(A-1)}$

DO

$B := B * b;$

$A := A - 1$

OD

$B = b^{a-A} \wedge A = 0 \longrightarrow B = b^a$

$\{ B = b^a \}$

Example 4

$\{\ \mathit{True}\ \}$

$X := x;$ $\quad X = [x_0; x_1; x_2 \ldots] \quad [x_1; x_2 \ldots] \quad [x_2 \ldots] \quad \ldots$

$Y := [];$ $\quad Y = [] \quad x_0 \# [] \quad x_1 \# x_0 \# [] \quad \ldots$

$(\mathit{rev}\ x) @ [] = \mathit{rev}\ x$

INV $\{\ (\mathit{rev}\ X) @ Y = \mathit{rev}\ x \}$

WHILE $X \neq []$

$(\mathit{rev}\ X) @ Y = \mathit{rev}\ x\ \wedge X \neq [] \longrightarrow$

$(\mathit{rev}\ (\mathit{tl}\ X)) @ ((\mathit{hd}\ X) \# Y) = \mathit{rev}\ x$

$= (\mathit{rev}\ X) @ Y$

$= (\mathit{rev}\ ((\mathit{hd}\ X) \# (\mathit{tl}\ X))) @ Y$

DO

$\quad Y := (\mathit{hd}\ X \# Y);$

$\quad X := \mathit{tl}\ X$

OD

$(\mathit{rev}\ X) @ Y = \mathit{rev}\ x\ \wedge X = []\ \longrightarrow Y = \mathit{rev}\ x$

$\{\ Y = \mathit{rev}\ x\ \}$

Example 5

Try with $b = 10 = 2^1 + 2^3$ or $b = 12 = 2^2 + 2^3$ (and e.g. a=3)

$\{ a \geq 0 \wedge b \geq 0 \}$

$A := a;\ B := b;\ C := 1;$ $\qquad a^b = 1 * a^b$

INV $\{ a^b = C * A^B \}$

WHILE $B \neq 0$ $\qquad a^b = C * A^B \wedge B \neq 0 \longrightarrow a^b = (C * A) * A^{B-1}$

DO

INV $\{ a^b = C * A^B \}$

WHILE (B mod 2 = 0)

$a^b = C * A^B \wedge B\ mod\ 2 = 0 \longrightarrow a^b = C * (A * A)^{B\ div\ 2}$

DO

$A := A * A;$

$B := B\ div\ 2;$

OD

$C := C * A;$

$B := B - 1$

OD $\qquad a^b = C * A^B \wedge B = 0 \longrightarrow C = a^b$

$\{ C = a^b \}$

Example 6

$LEQ\ A\ n = \forall k.\ k < n \longrightarrow\ A!k \leq piv$
$GEQ\ A\ n = \forall k.\ n < k < length\ A \longrightarrow A!k \geq piv$
$EQ\ A\ n\ m = \forall k.\ n \leq k \leq m \longrightarrow A!k = piv$

$\{\ 0 < length\ A\ \}$
$l := 0;\ u := length\ A - 1;\ A := a$
INV $\{\ LEQ\ A\ l \wedge GEQ\ A\ u \wedge u < length\ A \wedge l \leq length\ A \wedge A$ permutes $a\}$
WHILE $l \leq u$
DO
INV $\{\ LEQ\ A\ l \wedge GEQ\ A\ u \wedge u < length\ A \wedge l \leq length\ A \wedge A$ permutes $a\}$
WHILE $l < length\ A \wedge A!l \leq piv$ DO $l := l + 1$ OD;

INV $\{\ LEQ\ A\ l \wedge GEQ\ A\ u \wedge u < length\ A \wedge l \leq length\ A \wedge A$ permutes $a\}$
WHILE $0 < u \wedge piv \leq A!u$ DO $u := u - 1$ OD;

IF $l \leq u$ THEN $A := A[l := A!u, u := A!l]$ ELSE SKIP FI
OD
$\{\ LEQ\ A\ u \wedge EQ\ A\ u\ l \wedge GEQ\ A\ l \wedge A$ permutes $a\ \}$

Example 7

Reminder:

datatype ref = Ref int | Null
Pointer access: p→field
Pointer update: p→field :== v

Definition:

"*List nxt p Ps*" is a linked list, starting at pointer *p* following the next pointer through the function *nxt*, and where *Ps* contains the list of the pointers of the linked list.

$\{ \mathit{List\ nxt\ p\ Ps} \wedge X \in Ps \}$
INV $\{ \exists Qs.\ \mathit{List\ nxt\ p\ Qs} \wedge X \in Qs \}$
WHILE $p \neq \mathit{Null} \wedge p \neq \mathit{Ref}\ X$
DO
$\quad p := p \rightarrow nxt;$
OD
$\{ p = \mathit{Ref}\ X \}$

$\exists Qs.\ \mathit{List\ nxt\ p\ Qs} \wedge X \in Qs$

$\exists Qs.\ \mathit{List\ nxt\ p\ Qs} \wedge X \in Qs$
$\wedge p \neq \mathit{Null} \wedge p \neq \mathit{Ref}\ X \longrightarrow$
$\exists Qs.\ \mathit{List\ nxt}\ (p \rightarrow nxt)\ Qs \wedge X \in Qs$

$\exists Qs.\ \mathit{List\ nxt\ p\ Qs} \wedge X \in Qs$
$\wedge (p = \mathit{Null} \vee p = \mathit{Ref}\ X) \longrightarrow\ p = \mathit{Ref}\ X$

Example 8

What is is Isabelle function doing?

$fun\ f\ ::\ 'a\ list \Rightarrow' a\ list \Rightarrow' a\ list\ where$
$f\ []\ ys = ys|$
$f\ xs\ [] = xs|$
$f\ (x\#xs)\ (y\#ys) = x\#y\#\ f\ xs\ ys$

Example 8

What is is Isabelle function doing?

$fun\ \color{red}{splice}\ :: \ 'a\ list \Rightarrow' a\ list \Rightarrow' a\ list\ where$
$\quad \color{red}{splice}\ []\ ys = ys |$
$\quad \color{red}{splice}\ xs\ [] = xs |$
$\quad \color{red}{splice}\ (x \# xs)\ (y \# ys) = x \# y \#\ f\ xs\ ys$

Let's write it with linked lists!

Example 8

List nxt p Ps = *Path nxt p Ps Null*

Path nxt p Ps Null is a linked list from *p* to *q* following function *nxt* and containing list of pointers *Ps*

$\{$ *List nxt p Ps* $\wedge$ *List nxt q Qs* $\wedge$ (*set Ps* $\cap$ *set Qs*) $= \{\} \wedge$ *size Qs* $\leq$ *size Ps* $\}$

pp := *p*;

INV $\{$ $\exists$*PPs QQs PPPs*. *size QQs* $\leq$ *size PPs* $\wedge$
List nxt pp PPs $\wedge$ *List nxt q QQs* $\wedge$ *Path nxt p PPPs pp*
$\wedge$ *PPPs*@*splice PPs QQs* = *splice Ps Qs* $\wedge$
set PPs $\cap$ *set QQs* $= \{\} \wedge$ *distinct PPPs* $\wedge$ *set PPPs* $\cap$ (*set PPs* $\cup$ *set QQs*) $= \{\}$
$\}$

WHILE $q \neq$ *Null*

DO

qq := $q \rightarrow$ *nxt*; $q \rightarrow$ *nxt* := *pp* $\rightarrow$ *nxt*; *pp* $\rightarrow$ *nxt* = *q*; *pp* := $q \rightarrow$ *nxt*; *q* := *qq*;

OD

$\{$ *List nxt p* (*splice Ps Qs*) $\}$

Demo

COMP4161
Advanced Topics in Software Verification

A2 Recap

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

A2 Recap

Assignment 2 is now marked.

- Biggest frustration: *nat* arithmetic.
  - We know & sympathise.
  - Underflow conditions are tricky.
- Getting the right induction is *very* important.
  - A generalisation of the problem of the right invariant.
- The proofs about `to_nat` require helper lemmas.
  - e.g. `to_nat` $(2 * n)$, `to_nat` $((2 * n) + 1)$
- We will release a reference solution (or two) privately.

COMP4161
Advanced Topics in Software Verification

Exam Prep

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

Last Time

- ➜ The automated proof method **wp**
- ➜ The C Parser and translating C into Simpl
- ➜ AutoCorres and translating Simpl into monadic form
- ➜ The option and exception monads

Exam

- ➜ 24h take-home exam (same as previous years)

- ➜ Open book: can use any passive resource (books, slides, google, etc)
- ➜ **Not** allowed to ask for help from anyone
- ➜ **Not** allowed AI assistance for technical support (e.g. ChatGPT).
- ➜ starts 8am AEST, Thursday 28th Nov 2024, ends 7:59am AEST, Friday 29th Nov 2024

- ➜ Should be doable in about 4-6 hours.
  The 24h are for flexibility not for you to stay awake actual 24 hours.
- ➜ Recommend to start early, finish the easy questions first.
- ➜ Take breaks. Don't forget to eat :-)
- ➜ If there are clarification questions, make **private** threads on Ed.

Content

- Foundations & Principles
  - Intro, Lambda calculus, natural deduction [1,2]
  - Higher Order Logic, Isar (part 1) [2,3[a]]
  - Term rewriting [3,4]
- Proof & Specification Techniques
  - Inductively defined sets, rule induction [4,5]
  - Datatype induction, primitive recursion [5,7]
  - General recursive functions, termination proofs [7]
  - Proof automation, Isar (part 2) [8[b]]
  - Hoare logic, proofs about programs, invariants [8,9]
  - C verification [9,10]
  - Practice, questions, exam prep [10[c]]

[a] a1 due; [b] a2 due; [c] a3 due

COMP4161
Advanced Topics in Software Verification

λ

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have learned so far...

- ➜ λ calculus syntax
- ➜ free variables, substitution
- ➜ β reduction
- ➜ α and η conversion
- ➜ β reduction is confluent
- ➜ λ calculus is very expressive (turing complete)
- ➜ λ calculus results in an inconsistent logic

COMP4161
Advanced Topics in Software Verification

$$\lambda^{\rightarrow}$$

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have learned so far...

- ➜ Simply typed lambda calculus: $\lambda^{\rightarrow}$
- ➜ Typing rules for $\lambda^{\rightarrow}$, type variables, type contexts
- ➜ β-reduction in $\lambda^{\rightarrow}$ satisfies subject reduction
- ➜ β-reduction in $\lambda^{\rightarrow}$ always terminates
- ➜ Types and terms in Isabelle

COMP4161
Advanced Topics in Software Verification

$\lambda^{\rightarrow}$ and HOL

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

What we have learned so far...

- natural deduction rules for $\wedge$, $\vee$, $\longrightarrow$, $\neg$, iff...
- proof by assumption, by intro rule, elim rule
- safe and unsafe rules

- indent your proofs! (one space per subgoal)
- prefer implicit backtracking (chaining) or *rule_tac*, instead of *back*
- *prefer* and *defer*
- *oops* and *sorry*

COMP4161
Advanced Topics in Software Verification

HOL

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have learned so far...

- ➜ Isar style proofs
- ➜ proof, qed
- ➜ assumes, shows
- ➜ fix, obtain
- ➜ moreover, ultimately
- ➜ forward, backward
- ➜ mixing proof styles

COMP4161
Advanced Topics in Software Verification

HOL

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have learned today ...

- ➜ Defining HOL
- ➜ Higher Order Abstract Syntax
- ➜ Deriving proof rules
- ➜ More automation
- ➜ Equations and Term Rewriting

COMP4161
Advanced Topics in Software Verification

$\longrightarrow$

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have seen today...

- ➜ Equations and Term Rewriting
- ➜ Confluence and Termination of reduction systems
- ➜ Term Rewriting in Isabelle

COMP4161
Advanced Topics in Software Verification

$\longrightarrow$

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have learned today ...

- ➜ Conditional term rewriting
- ➜ Congruence rules
- ➜ AC rules
- ➜ More on confluence

COMP4161 Advanced Topics in Software Verification

{}

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have learned today ...

➜ Sets
➜ Type Definitions
➜ Inductive Definitions

COMP4161
Advanced Topics in Software Verification

{}

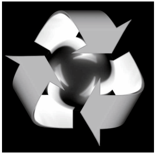
Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have learned today ...

➜ Formal background of inductive definitions
➜ Definition by intersection
➜ Computation by iteration
➜ Formalisation in Isabelle

COMP4161
Advanced Topics in Software Verification



Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have seen today ...

- ➜ Datatypes
- ➜ Primitive recursion
- ➜ Case distinction
- ➜ Structural Induction

COMP4161
Advanced Topics in Software Verification

fun

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have see today ...

- General recursion with **fun**/**function**
- Induction over recursive functions
- How **fun** works
- Termination, partial functions, congruence rules

COMP4161
Advanced Topics in Software Verification

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have seen today ...

- ➜ sledgehammer
- ➜ nitpick
- ➜ quickcheck

COMP4161
Advanced Topics in Software Verification

{P} . . . {Q}

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have seen today ...

- ➜ Syntax of a simple imperative language
- ➜ Operational semantics
- ➜ Program proof on operational semantics
- ➜ Hoare logic rules
- ➜ Soundness of Hoare logic

COMP4161
Advanced Topics in Software Verification

$$\{P\} \ldots \{Q\}$$

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have seen today ...

- ➜ Weakest precondition
- ➜ Verification conditions
- ➜ Example program proofs
- ➜ Arrays, pointers

COMP4161
Advanced Topics in Software Verification

$\gg=$

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

We have seen today

- ➜ Deep and shallow embeddings
- ➜ Isabelle records
- ➜ Nondeterministic State Monad with Failure
- ➜ Monadic Weakest Precondition Rules

COMP4161
Advanced Topics in Software Verification

C

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

Today we have seen

- ➜ The automated proof method **wp**
- ➜ The C Parser and translating C into Simpl
- ➜ AutoCorres and translating Simpl into monadic form
- ➜ The option and exception monads

Two Final Announcements

Two last things to mention before the end of the course.

Thanks for being a great audience!

- ➜ It's great to see the level of engagement and interesting questions in this course.
- ➜ Good luck with the remaining assignment and exam.
- ➜ There may be more opportunities to use Isabelle and other theorem provers in research & industry.

Provisional: TS Hiring

Thomas, Miki & Rob work for/with the TS group at UNSW.

TS has funding for some Isabelle and seL4 related projects:

- seL4 specification gap
- seL4 WCET
- seL4 time-protection extensions