Program Verification

Hoare rules mostly syntax directed, but:
• When to apply the rule of consequence?
• What invariant to use for while?
• How to prove implications (in conseq rule)?

The last one involves theorem proving:
  - Doable

Loop invariants are the hardest problem

Technique: Weakest Preconditions

Define \( \text{wp}(c, B) \) inductively on \( c \), following Hoare rules

\[
\text{wp}(c_1; c_2, B) = \begin{cases} 
\text{wp}(c_1, \text{wp}(c_2, B)) & \text{if } e \Rightarrow c_1 \land \neg e \Rightarrow c_2 \\
\text{wp}(x:=e, B) & \text{if } e \Rightarrow c_1 \land \neg e \Rightarrow c_2 \\
\text{wp}(\text{if } e \text{ then } c_1 \text{ else } c_2, B) & \text{if } e \Rightarrow c_1 \land \neg e \Rightarrow c_2 
\end{cases}
\]

After what preconditions does postcond. \( x>0 \) hold?

\( \text{WP}(c, B) \): weakest predicate s.t. \( \{ \text{WP}(c, B) \} c \{ B \} \)
• For any \( A \) we have \( \{ A \} c \{ B \} \) iff \( A \Rightarrow \text{WP}(c, B) \)

How to verify \( \{ A \} c \{ B \} \)?
1. Compute: \( \text{WP}(c, B) \)
2. Prove: \( A \Rightarrow \text{WP}(c, B) \)
Weakest Preconditions for Loops

Start from the equivalence
\[
\text{while } b \text{ do } c =
\begin{align*}
\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else } \text{skip}
\end{align*}
\]

Let \( W = \wp(\text{while } b \text{ do } c, B) \)

It must be that: \( W = (b \Rightarrow \wp(c, W) \land \neg b \Rightarrow B) \)

But this is a recursive equation!

• We’ll return to finding loop WPs later ...

Technique: Strongest Postconditions

\[
\begin{align*}
\{ y > 100 \} & \quad x := y \quad \{ x > 10 \}
\end{align*}
\]

\[
\begin{align*}
\{ y > 100 \} & \quad x := y \quad \{ x > 20 \}
\end{align*}
\]

\[
\begin{align*}
\{ y > 100 \} & \quad x := y \quad \{ x > 100 \}
\end{align*}
\]

What postcond. is guaranteed after prec. \( y > 100 \) ?

\( \text{SP}(c, A) \): strongest predicate s.t. \( \{ B \} \ c \ \{ \text{SP}(c, A) \} \)

• For any \( B \) we have \( \{ A \} \ c \ \{ B \} \iff \text{SP}(c, A) \Rightarrow B \)

How to verify \( \{ A \} \ c \ \{ B \} \)?

1. Compute: \( \text{SP}(c, A) \)
2. Prove: \( \text{SP}(c, A) \Rightarrow B \)

Strongest Postconditions

Define \( \text{sp}(c, B) \) inductively on \( c \), following Hoare rules

\[
\begin{align*}
\text{sp}(c_1; c_2, A) &= \quad \vdash [A] c_1 \ [B] \quad \vdash [B] c_2 \ [C] \\
\text{sp}(c, A) &= \quad \vdash [A] \quad \exists x_0. [x_0/x]A \land x=[x_0/x]e \\
\text{sp}(if \ e \ then \ c_1 \ else \ c_2, A) &= \quad \vdash [A \land b] c_1 \ [B] \quad \vdash [A \land \neg b] c_2 \ [B] \\
\end{align*}
\]

Axiomatic Semantics over Flow Graphs

“Relaxing” specifications via rule of consequence
Sequential Composition

\[
\begin{align*}
& \text{Backwards using weakest preconditions} \\
& \text{Forwards using strongest postconditions}
\end{align*}
\]

Conditionals

\[
\begin{align*}
& \text{Forwards} \\
& \text{Backwards}
\end{align*}
\]

Joins

\[
\begin{align*}
& \text{Forwards} \\
& \text{Backwards}
\end{align*}
\]

Conditional+Join: Forward

\[
\begin{align*}
& \text{Check the implications (simplifications)}
\end{align*}
\]
**Conditionals+Joins: Backward**

\[
\{ (x \neq 0 \land true) \lor (x = 0 \land a = 2^x) \}
\]

\[
\{ 2^x = 2^x \} \quad \text{F}
\]

\[
\{ a := 2^x \}
\]

\[
\{ a = 2^x \}
\]

---

**Forward or Backward?**

- **Forward reasoning**
  - Know the precondition
  - Want to know what postcond the code guarantees

- **Backward reasoning**
  - Know what we want to code to establish
  - Want to know under what preconditions this happens

---

**Another Example: Double Locking**

“An attempt to re-acquire an acquired lock or release a released lock will cause a **deadlock.**”

Calls to **lock** and **unlock** must **alternate.**

---

**Locking Rules**

Boolean variable **locked** = lock is held or not

- \{¬locked \land P[true/locked] \} **lock** { P }
  - **lock** behaves as **assert(!locked); locked:=true**

- \{ locked \land P[false/locked] \} **unlock** { P }
  - **unlock** behaves as **assert(locked); locked:=false**
**What about real languages?**

- Loops
- Function calls
- Pointers

**Reasoning about loops: Rules**

\[ \vdash \{ A \land b \} \ c \ { A } \]

\[ \vdash \{ A \} \ while \ b \ do \ c \ \{ A \land \neg b \} \]

Rewrite A with I: Loop Invariant

\[ \vdash \{ I \land b \} \ c \ \{ I \} \]

\[ P \Rightarrow I \]

\[ \vdash \{ I \} \ while \ b \ do \ c \ \{ I \land \neg b \} \]

\[ I \land \neg b \Rightarrow Q \]

\[ \vdash \{ P \} \ while \ b \ do \ c \ \{ Q \} \]

Rule of Consequence

**Review**

\[ x := E \]

\[ \{ P \} \]

\[ \{ P \} \]

if \( Q \Rightarrow P[E \backslash x] \}

\[ \{ P \} \]

if \( P_1 \Rightarrow P \) and \( P_2 \Rightarrow P \)

\[ \{ P_1 \} \]

\[ \{ P_2 \} \]

if \( P \land E \Rightarrow P_1 \)

\[ \{ P \} \]

if \( P \land \neg E \Rightarrow P_2 \)

Implication is always in the direction of the control flow
Reasoning about loops: Flow Graphs

- Loops can be handled using conditionals and joins
- Consider the while b do S statement

$$\begin{align*}
\text{while } &b \text{ do } S
\end{align*}$$

Loop invariant

- if \( P \Rightarrow I \) (loop invariant holds initially)
- and \( I \land \neg b \Rightarrow Q \) (loop establishes the postcondition)
- and \( \{ I \land b \} S \{ I \} \) (loop invariant is preserved)

Loop Example

Verify:

\{ x=8 \land y=16 \} while(x>0) \{ x--; y=2; \} \{ y=0 \}

Find an appropriate invariant \( I \)
- Holds initially \( x = 8 \land y = 16 \)
- Holds at the end \( y == 0 \)

Loop Example (II)

Guess invariant \( y = 2*x \)

Check:
- Initial: \( x = 8 \land y = 16 \Rightarrow y = 2*x \)
- Preservation: \( y = 2*x \land x>0 \Rightarrow y-2 = 2*(x-1) \)
- Final: \( y = 2*x \land x \leq 0 \Rightarrow y = 0 \) Invalid

Loop Example (III)

Guess invariant \( y = 2*x \land x \geq 0 \)

Check:
- Initial: \( x = 8 \land y = 16 \Rightarrow y = 2*x \land x \geq 0 \)
- Preservation:
  \( y = 2*x \land x \geq 0 \land x>0 \Rightarrow y-2 = 2*(x-1) \land x-1 \geq 0 \)
- Final: \( y = 2*x \land x \geq 0 \land x \leq 0 \Rightarrow y = 0 \)
Loops Discussion

• Simple forward/backward propagation fails
• Require loop invariants
  - Hardest part of program verification
  - Guess the invariants (existing programs)
  - Write the invariants (new programs)

Note:
Invariant depends on what you want to prove!

Verification Example

int square(int n) {
    int k=0, r=0, s=1;
    { true }
    while(k != n) {
        r = r + s;
        s=s+2;
        k=k + 1;
        { r=0 ∧ k=0 ∧ s=1 }
        \[ \text{Pick I: } r = k^2 \]
        \[ \text{I : } \{ r=k^2 \} \]
    }
    return r;
    \[ \{ r=0 ∧ k=0 \} \]
}

Need: \[ \{ r=k^2 ∧ s=2k+1 ∧ \ldots \} \] \[ \{ r=k^2 ∧ s=2k+1 \} \]
i.e. \[ \{ r=k^2 ∧ s=2k+1 \} \Rightarrow WP(c, \{ r=k^2 \}) \]
i.e. \[ \{ r=k^2 ∧ s=2k+1 \} \Rightarrow \] \[ \{ r+s=(k+1)^2 \} \] Valid

What about real languages?

• Loops
• Function calls
• Pointers
**Functions are big instructions**

Suppose we have verified \( bsearch \)

```c
int bsearch(int a[], int p) {
    { sorted(a) } // Precondition
    ... 
    { r=-1 \lor (r \geq 0 \land r < a.length \land a[r]=p)}
    return r;
}
```

Function spec = precondition + postcondition
- Also called a contract

**Function Calls**

- Consider a call to function \( y:=f(int \ E) \)
  - return variable \( r \)
  - precondition Pre, postcondition Post

  Rule for function call:

  \[
  \vdash P \Rightarrow Pre[E/x] \quad \vdash Pre \quad \vdash Post[E/x, y/r] \Rightarrow Q \\
  \vdash [P] y:=f(E) [Q]
  \]

**Function Call: Example**

Consider the call

```c
int bsearch(int a[],int p) {
    { sorted(a) }
    ... 
    { r=-1 \lor (r \geq 0 \land r<a.length \land a[r]=p)}
    return r;
}
```

- sorted[array] \( \Rightarrow \) Pre[a := arr]
- Post[y/r,arr/a, 5/p] \( \Rightarrow \) (y=-1 \lor arr[y]=5)
What about real languages?

- Loops
- Function calls
- Pointers

Assignment and Aliasing

Does assignment rule work with aliasing?
If \( *x \) and \( *y \) are aliased then:
\[
\{ x = y \} \ x := 5 \ \{ *x + *y = 10 \}
\]

Hoare Rules: Assignment and References

- When is the following Hoare triple valid?
  \[
  \{ A \} \ x := 5 \ \{ *x + *y = 10 \}
  \]
- A should be “\( *y = 5 \) or \( x = y \)”

- but Hoare rule for assignment gives:
  \[
  [5/*x]*(x + *y = 10)
  = 5 + *y = 10
  = *y = 5
  \]
  (uh oh! we lost one case! What gives?)

Hoare Rules: Assignment and References

- Modeling writes with memory expressions
- Treat memory as a whole w/ memory variables \( (M) \)
- \( \text{upd}(M, E_1, E_2) \) : update \( M \) at addr \( E_1 \) with value \( E_2 \)
- \( \text{sel}(M, E_1) \) : read \( M \) at address \( E_1 \)

Reason about memory expressions with McCarthy’s rule

\[
\begin{array}{ll}
\text{sel}(\text{upd}(M, E_1, E_2), E_3) = & E_2 \quad \text{if } E_1 = E_3 \\
& \text{sel}(M, E_3) \quad \text{if } E_1 \neq E_3
\end{array}
\]

Assignment (update) changes the value of memory

\[
\{ B[\text{upd}(M, E_1, E_2)/M] \} \ E_1 := E_2 \ \{ B \}
\]
Memory Aliasing

- Consider again: \{A\} *x:=5 \{*x+y=10 \}
- We obtain:
  \[ A = [\text{upd}(M, x, 5)/M] (*x+y=10) \]
  \[ = [\text{upd}(M, x, 5)/M] (\text{sel}(M, x) + \text{sel}(M, y) = 10) \]
  \[ = \text{sel}(\text{upd}(M, x, 5), x) + \text{sel}(\text{upd}(M, x, 5), y) = 10 \]
  \[ = 5 + \text{sel}(\text{upd}(M, x, 5), y) = 10 \]
  \[ = \text{if } x = y \text{ then } 5 + 5 = 10 \text{ else } 5 + \text{sel}(M, y) = 10 \]
  \[ = x = y \text{ or } *y = 5 \]

Algorithmic Program Verification

...or how does ESC/Java work ?

Q: How to algorithmically prove \{P\} c \{Q\} ?

If no loops:
1. Compute: WP(c, Q)
2. Prove: P \Rightarrow WP(c, Q)

Verification Condition
“Discharged” using Auto. Theorem Prover

Program Verification Tools

- Semi-automated
  - You write some invariants and specifications
  - Tool tries to fill in the other invariants
  - And to prove all implications
  - Explains when implication is invalid: counterexample for your specification

- ESC/Java
- Spec#b

VC Generation for Loops

Suppose all loops annotated with Invariant
\[ \text{while}_I \ b \ \text{do } c \]

Again, lets compute a VC such that:
if VC is valid (true) then \{P\} c \{Q\}

Q: Why not iff ?

as the loop invariants may be bogus...
VCGen

We will write a function VCG:
\[
\text{VCG: comm} \times (\text{pred} \times \text{pred list}) \rightarrow (\text{pred} \times \text{pred list})
\]

Suppose \((Q', L') = \text{VCG}(c, (Q, L))\)
Then VC for \([P] c \{Q\} \)
\[
\text{is: } P \Rightarrow Q' \land \{f \text{ in } L'\}
\]

- \(L'\): the set of conditions that must be true
  - From loops (init, preservation, final)
- \(Q'\): “precondition” modulo invariants...

ESC/Java

Semi-automated
- You write the invariants
- ESC/Java:
  - VCGen:
    - Simplify: Theoremprover to prove VC
- Explains when implication is invalid:
  counterexample for your specification