# Computing Weakest Preconditions

- Idea: compute WP(*P*,B) recursively according to the structure of the program *P.*
- Problem: How to deal with loops?
- Solution:
  - introduce loop-free intermediate language
  - translate program to simplified program in the intermediate language
  - then compute WP on simplified program.

# Loop-Free Guarded Commands

- Introduce loop-free guarded commands as an intermediate representation of the verification condition
- $c$ ::= assume b | block if b does not hold

| | |
|---|---|
| $c$ ::= assume b | block if b does not hold |
| \| assert b | fail if b does not hold |
| \| havoc x | nondet. assignment |
| \| $c_1$ ; $c_2$ | sequencing |
| \| $c_1 \square c_2$ | nondet. choice |

# From Programs to Guarded Commands

- GC(`skip`) =
- GC($x$ := $e$) =
- GC($c_1$ ; $c_2$) =
- GC(`if` $b$ `then` $c_1$ `else` $c_2$) =
- GC({I} `while` $b$ `do` $c$) =

# From Programs to Guarded Commands

- GC(`skip`) =

  assume true
- GC($x$ := $e$) =

  havoc *tmp*; assume *tmp* = $x$; where *tmp* is fresh

  havoc $x$; assume ($x$ = $e[tmp/x]$)
- GC($c_1$ ; $c_2$) =

  GC($c_1$) ; GC($c_2$)
- GC(`if` $b$ `then` $c_1$ `else` $c_2$) =

  (assume $b$; GC($c_1$)) □ (assume $\neg b$; GC($c_2$))
- GC({I} `while` $b$ do $c$) = ?

# Guarded Commands for Loops

- GC({I} while *b* do *c*) =

  assert I;

  havoc $x_1$; …; havoc $x_n$;

  assume I;

  (assume *b*; GC(*c*); assert I; assume false) □

  assume ¬*b*

where $x_1$, …, $x_n$ are the variables assigned in *c*

# Computing Weakest Preconditions

- WP(assume $b$, B) =
- WP(assert $b$, B) =
- WP(havoc $x$, B) =
- WP($c_1$;$c_2$, B) =
- WP($c_1$ □ $c_2$,B) =

# Computing Weakest Preconditions

- WP(assume $b$, B) = $b \Rightarrow$ B
- WP(assert $b$, B) = $b \wedge$ B
- WP(havoc $x$, B) = B[$a/x$]     ($a$ fresh in B)
- WP($c_1;c_2$, B) = WP($c_1$, WP($c_2$, B))
- WP($c_1 \square c_2$,B) = WP($c_1$, B) $\wedge$ WP($c_2$, B)

# Computing Weakest Preconditions

- WLP(assume *b*, B) = $b \Rightarrow$ B
- WLP(assert *b*, B) = $b \Rightarrow$ B
- WLP(havoc *x*, B) = B[*a*/*x*] (*a* fresh in B)
- WLP($c_1$;$c_2$, B) = WP($c_1$, WP($c_2$, B))
- WLP($c_1 \; \square \; c_2$,B) = WP($c_1$, B) $\wedge$ WP($c_2$, B)

# Putting Everything Together

- Given a Hoare triple $H \equiv \{A\}\ P\ \{B\}$
- Compute $c_H$ = assume A; GC(P); assert B
- Compute $VC_H = WP(c_H, \text{true})$
- Infer $\vdash VC_H$ using a theorem prover.

# Example: VC Generation

$\{n \geq 0\}$

```
p := 0;
x := 0;
{p = x * m ∧ x ≤ n}
while x < n do
    x := x + 1;
    p := p + m
{p = n * m}
```

# Example: VC Generation

- Computing the guarded command

```
assume n ≥ 0;
GC(   p := 0;
      x := 0;
      {p = x * m ∧ x ≤ n}
      while x < n do
          x := x + 1;
          p := p + m  );
assert p = n * m
```

# Example: VC Generation

- Computing the guarded command

assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

GC( $x := 0$;

$\{p = x * m \wedge x \leq n\}$

while $x < n$ do

$x := x + 1$;

$p := p + m$ );

assert $p = n * m$

# Example: VC Generation

- Computing the guarded command

assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

GC( $\{p = x * m \land x \leq n\}$

while $x < n$ do

$x := x + 1$;

$p := p + m$ );

assert $p = n * m$

# Example: VC Generation

- Computing the guarded command

assume $n \geq 0$;<br>
assume $p_0 = p$; havoc $p$; assume $p = 0$;<br>
assume $x_0 = x$; havoc $x$; assume $x = 0$;<br>
assert $p = x * m \wedge x \leq n$;<br>
havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$;<br>
  (assume $x < n$;<br>
   GC( $x := x + 1$;<br>
        $p := p + m$);<br>
   assert $p = x * m \wedge x \leq n$; assume false )<br>
□ assume $x \geq n$;<br>
assert $p = n * m$

# Example: VC Generation

- Computing the guarded command

assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$;

havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$;

(assume $x < n$;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc $p$; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$; assume false)

 assume $x \geq n$;

assert $p = n * m$

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \land x \leq n$;

havoc $x$; havoc $p$; assume $p = x * m \land x \leq n$;

(assume $x < n$;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc $p$; assume $p = p_1 + m$;
assert $p = x * m \land x \leq n$; assert false)

□ assume $x \geq n$;

assert $p = n * m$, true)

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$;

(assume $x < n$;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc $p$; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$; assume false)

 assume $x \geq n$, $p = n * m$)

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$,

(WP( (assume $x < n$;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc $p$; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$; assume false)) $\Rightarrow$
$p = n * m$)

$\wedge$ ($x \geq n \Rightarrow p = n * m$)))

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$,

(WP( (assume x < n;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc $p$; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$), false $\Rightarrow p = n * m$)

$\wedge$ (x $\geq$ n $\Rightarrow p = n * m$)))

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$,

(WP( (assume x < n;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$;

assume $p_1 = p$; havoc $p$; assume $p = p_1 + m$;

assert $p = x * m \wedge x \leq n$), true)

$\wedge$ ($x \geq n \Rightarrow p = n * m$)))

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$,

(WP( (assume $x < n$;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$;

assume $p_1 = p$; havoc $p$),

$p = p_1 + m \Rightarrow p = x * m \wedge x \leq n$)

$\wedge$ ($x \geq n \Rightarrow p = n * m$)))

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$,

(WP( (assume $x < n$;

assume $x_1 = x$; havoc $x$; assume $x = x_1 + 1$),

$p_1 = p \wedge pa_1 = p_1 + m \Rightarrow pa_1 = x * m \wedge x \leq n$)

$\wedge (x \geq n \Rightarrow p = n * m)))$

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \land x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \land x \leq n$,

(WP( (assume x < n),

$(x_1 = x \land xa_1 = x_1 + 1 \land$
$p_1 = p \land pa_1 = p_1 + m) \Rightarrow pa_1 = x * m \land x \leq n)$

$\land$ (x $\geq$ n $\Rightarrow p = n * m$)))

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

WP(havoc $x$; havoc $p$; assume $p = x * m \wedge x \leq n$,

$(x < n \wedge$

$x_1 = x \wedge xa_1 = x_1 + 1 \wedge$

$p_1 = p \wedge pa_1 = p_1 + m) \Rightarrow pa_1 = x * m \wedge x \leq n)$

$\wedge (x \geq n \Rightarrow p = n * m)))$

# Example: VC Generation

- Computing the weakest precondition

WP ( assume $n \geq 0$;

assume $p_0 = p$; havoc $p$; assume $p = 0$;

assume $x_0 = x$; havoc $x$; assume $x = 0$;

assert $p = x * m \wedge x \leq n$,

$(pa_2 = xa_2 * m \wedge xa_2 \leq n \wedge$

$xa_2 < \mathrm{n} \wedge$

$x_1 = xa_2 \wedge xa_1 = x_1 + 1 \wedge$

$p_1 = pa_2 \wedge pa_1 = p_1 + m) \Rightarrow pa_1 = xa_2 * m \wedge xa_2 \leq n)$

$\wedge (\mathrm{x} \geq \mathrm{n} \Rightarrow p = n * m)))$

# Example: VC Generation

- Computing the weakest precondition

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \Rightarrow$$
$$pa_3 = xa_3 * m \wedge xa_3 \leq n \wedge$$
$$(pa_2 = xa_2 * m \wedge xa_2 \leq n \wedge$$
$$xa_2 < n \wedge$$
$$x_1 = xa_2 \wedge xa_1 = x_1 + 1 \wedge$$
$$p_1 = pa_2 \wedge pa_1 = p_1 + m) \Rightarrow pa_1 = xa_2 * m \wedge xa_2 \leq n)$$
$$\wedge (x \geq n \Rightarrow p = n * m)))$$

# Example: VC Generation

- The resulting VC is equivalent to the conjunction of the following implications

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \Rightarrow$$
$$pa_3 = xa_3 * m \wedge xa_3 \leq n$$

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \wedge pa_2 = xa_2 * m \wedge xa_2 \leq n \Rightarrow$$
$$xa_2 \geq n \Rightarrow pa_2 = n * m$$

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \wedge pa_2 = xa_2 * m \wedge xa_2 < n \wedge x_1 = xa_2 \wedge xa_1 = x_1 + 1 \wedge p_1 = pa_2 \wedge pa_1 = p_1 + m \Rightarrow$$
$$pa_1 = xa_1 * m \wedge xa_1 \leq n$$

# Example: VC Generation

- simplifying the constraints yields

$$n \geq 0 \Rightarrow 0 = 0 * m \wedge 0 \leq n$$

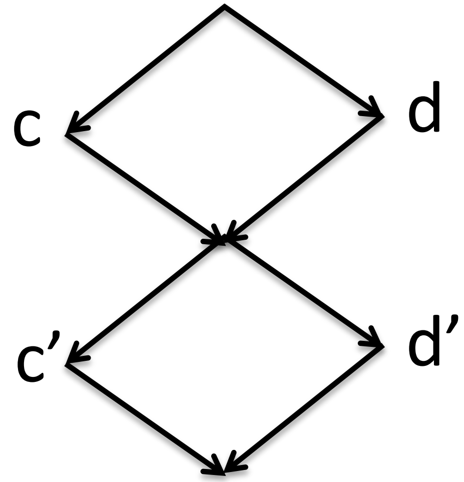
$$xa_2 \leq n \wedge xa_2 \geq n \Rightarrow xa_2 * m = n * m$$

$$xa_2 < n \Rightarrow xa_2 * m + m = (xa_2 + 1) * m \wedge xa_2 + 1 \leq n$$

- all of these implications are valid, which proves that the original Hoare triple was valid, too.

# The Diamond Problem

assume A;
*c* □ *d*;
*c′* □ *d′*;
assert B




$$A \Rightarrow WP(c, WP(c', B) \wedge WP(d', B)) \wedge$$
$$WP(d, WP(c', B) \wedge WP(d', B))$$

- Number of paths through the program can be exponential in the size of the program.
- Size of weakest precondition can be exponential in the size of the program.

# Avoiding the Exponential Explosion

Ideas?

# Avoiding the Exponential Explosion

Ideas?

1. Introduce propositional variables that stand for repeated subformulas
   - yields formulas that are linear in the program size
   - burden has now shifted to the theorem prover (often still exponential behavior)
2. Remove redundancies from the VCs entirely
   - yields formula that are quadratic in the program size
   - usually more efficient once theorem prover is factored in

# Removing Redundancy from VCs

- The following equivalence holds for arbitrary programs c and formulas B:

$$WP(c, B) \equiv WP(c, true) \wedge WLP(c, B)$$

- We got rid of B below WP. Can we also get rid of B below WLP?

# Passive Guarded Commands

- $c$ ::= assume b — block if b does not hold
  - | assert b — fail if b does not hold
  - ~~| havoc x — nondet. assignment~~
  - | $c_1 ; c_2$ — sequencing
  - | $c_1 \square c_2$ — nondet. choice

Passive programs are also often said to be in static single assignment (SSA) form. For loop-free programs, the SSA form can be obtained using a simple program transformation.

# Removing Redundancy from VCs

- The following equivalence holds for arbitrary programs c and formulas B:

$$\mathrm{WP}(c, B) \equiv \mathrm{WP}(c, \mathit{true}) \wedge \mathrm{WLP}(c, B)$$

- For passive programs c we also have:

$$\mathrm{WLP}(c, B) \equiv \mathrm{WLP}(c, \mathit{false}) \vee B$$

# Removing Redundancy from VCs

- Using the equations from the previous slides, we can compute WP for passive programs recursively according to the following equation:

$$\mathrm{WP}(c, B) \equiv \mathrm{WP}(c, \mathrm{true}) \wedge (\mathrm{WLP}(c, \mathrm{false}) \vee B)$$

- WP(c, B) is now quadratic in the size of c
- There is no duplication of B for each path in c

# Translating Method Calls to GCs

```
method m (p₁: T_1, ..., pₖ: Tₖ) returns (r: T)
    requires P
    modifies x₁, ..., xₙ
    ensures Q
```

A method call

```
  y := y₀.m(y₁, ..., yₖ);
```

is desugared into the guarded command

assert $P[y_0/\mathtt{this}, y_1/p_1, \ldots, y_k/p_k]$;

havoc $x_1$; ..., havoc $x_n$; havoc $y$;

assume $Q[y_0/\mathtt{this}, y_1/p_1, \ldots, y_k/p_k, y/r]$

# Handling More Complex Program State

- When is the following Hoare triple valid?

  {A} x.f := 5 {x.f + y.f = 10}

- A ought to imply "y.f = 5 $\vee$ x = y"
- The IMP Hoare rule for assignment would give us:

  (x.f + y.f = 10) [5/x.f]

  $\equiv$ 5 + y.f = 10

  $\equiv$ y.f = 5 (we lost one case)

- How come the rule does not work?

# Modeling the Heap

- We cannot have side-effects in assertions
  - While generating the VC we must remove side-effects!
  - But how to do that when lacking precise aliasing information?
- Simple solution: postpone alias analysis to the theorem prover
- Model the state of the heap as a symbolic mapping from addresses to values:
  - If e denotes an address and h a heap state then:
  - sel(h,e) denotes the contents of the memory cell
  - upd(h,e,v) denotes a new heap state obtained from h by writing v at address e

# Heap Models

- We allow variables to range over heap states
  - So we can quantify over all possible heap states.
- Model 1
  - One "heap" for each object
  - One index constant for each field (we postulate $f_1 \neq f_2$).
  - r.f is sel(r,f) and r.f := e is r := upd(r, f, e)
- Model 2 (Burstall-Bornat)
  - One "heap" for each field
  - The object address is the index
  - r.f is sel(f,r) and r.f := e is f := upd(f,r,e)