CS 477: Hoare Logic

Sasa Misailovic

Based on previous slides by Elsa Gunter, which were based on earlier slides by Gul Agha, and Mahesh Viswanathan

University of Illinois at Urbana-Champaign

Axiomatic Semantics

- Goal: Derive statements of form

$$\{P\}\ C\ \{Q\}$$

  - P , Q logical statements about state,
    P precondition,
    Q postcondition,
    C program

- Example: $\{x > 1\}\ x := x + 1\ \{x > 2\}$

Axiomatic Semantics

- An expression {P} C {Q} is a *partial correctness* statement
- For *total correctness* must also prove that C terminates (i.e. doesn't run forever)
  - Written: [P] C [Q]
- Will only consider partial correctness here

All Rules on One Slide

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \qquad \{P \text{ and (not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Precondition Strengthening

$$\frac{P \rightarrow P' \qquad \{P'\}\ C\ \{Q\}}{\{P\}\ C\ \{Q\}}$$

Postcondition Weakening

$$\frac{\{P\}\ C\ \{Q'\} \qquad Q' \rightarrow Q}{\{P\}\ C\ \{Q\}}$$

Rule of Consequence

$$\frac{P \rightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \rightarrow Q}{\{P\}\ C\ \{Q\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

Approaches in the Ideal World

- Strongest postcondition
  - Need to worry about existential quantifiers

    $sp(x:=e, Pre) = \exists x' .\ \ Pre[x'=x] \ \wedge\ x = (\ e[x'/x]\ )$
  - Captures the new and all previous values of x

Approaches in the Ideal World

- Weakest precondition (by Dijkstra)
  - No need to worry about existential quantifiers here:

$$wp(x:=e, Post) = Post[e/x]$$

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

  - Sequence and conditionals are handled also "nicely"
  - But loops pose problems!
  - Need to infer the invariant from the postcondition, and reason about total correctness (loop *variant*)

Verification Condition Generation

The Assignment Rule

$$\frac{}{\{P\,[e/x]\}\ x := e\ \{P\}}$$

If Then Else

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\} \quad \{P \text{ and (not } B)\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ \{Q\}}$$

Sequencing

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1;\ C_2\ \{R\}}$$

While

$$\frac{\{\ P \text{ and } B\ \}\ \ C\ \ \{\ P\ \}}{\{\ P\ \}\ \textbf{while}\ \ B\ \ \textbf{do}\ \ C\ \textbf{od}\ \ \{\ P \text{ and not } B\ \}}$$

- VC : Stmt x Predicate -> Predicate
- From the statement and postcondition infer precondition

- VC(x := e, Q) = Q [e/x]
- VC(C1; C2 , Q) = VC (C1, VC (C2, Q))
- VC(if b C1 else C2, Q) = $b \Rightarrow VC(C1, Q) \wedge \neg b \Rightarrow VC(C2, Q)$

  (alternative:) $(b \wedge VC(C1, Q)) \vee (\neg b \wedge VC(C2, Q))$
-

Verification Condition Generation: While

$$\frac{\{\,P \text{ and } B\,\}\ C\ \{\,P\,\}}{\{\,P\,\}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ \{\,P \text{ and not } B\,\}}$$

- While loop: let us annotate it with the invariant first

- We want to get a VC(while b C, Q)
  - Inv is initially true
  - The triple {Inv $\wedge$ b} C {Inv} is valid in every iteration
  - which means Inv $\wedge$ b $\Rightarrow$ VC (C, Inv) for all variable assignments
  - Inv $\wedge \neg$b $\Rightarrow$ Q

- Bring it together:

$$\text{Inv} \wedge (\forall x1 \ldots xn\ .\ (\text{Inv} \wedge b \Rightarrow \text{VC}\ (C, \text{Inv})) \wedge (\text{Inv} \wedge \neg b \Rightarrow Q))$$

- Can simplify:
  (exercise) $\text{Inv} \wedge (\forall x1 \ldots xn\ .\ \text{Inv} \Rightarrow ((b \Rightarrow \text{VC}\ (C, \text{Inv})) \wedge (\neg b \Rightarrow Q)))$

How do we know we defined the triples the right way?

- Let us connect the triples with the operational semantics
- First, we define the semantics of assertions
- Then, we define soundness and completeness of Hoare Triples

Semantics of Assertions

- A is a first-order formula (terms t)
- $s \models A$: the assertion holds in state s
- Define satisfaction inductively (e.g. fragment with $\wedge$ and $\forall$) :

$$\frac{(t1,s) \Downarrow v1 \quad (t2,s) \Downarrow v2 \quad v1 \text{ rop } v2}{s \models t1 \text{ rop } t2}$$

$$\frac{s \models A \quad s \models B}{s \models A \wedge B} \qquad \frac{\forall\, v\,.\, s\,[v/x] \models A}{s \models \forall x.\, A}$$

Soundness of Hoare Triples

- Define partial correctness via operational semantics:
- Recall: "if the program satisfies predicate P before executing the statement C, then after its execution finishes, the program will satisfy predicate Q."

$$\{P\}\ C\ \{Q\}$$

$$\Longrightarrow$$

$$\forall s_1 \forall s_2 .\ \ (\ s_1 \models P \wedge (C,s_1) \Downarrow s_2\ ) \ \Longrightarrow\ \ s_2 \models Q$$

- This can be proved with the usual induction by cases. That justifies our syntactic proofs on {P}C{Q}

Completeness of Hoare Triples

- We should be able to prove true statements:

$$\forall s_1 \forall s_2 .\ \ (s_1 \models P \wedge (C,s_1) \Downarrow s_2) \ \Rightarrow\ s_2 \models Q$$

$$\Rightarrow$$

$$\{P\}\ C\ \{Q\}$$

- Roadblock: Proving can only be complete relative to the logic (Goedels incompleteness theorem):
  "Any consistent formal system *F* within which a <u>certain amount of elementary arithmetic can be carried out is incomplete</u>; i.e., there are statements of the language of *F* which can neither be proved nor disproved in *F*" (Raatikainen 2015)
- Reduction to our case: $\models Q$ iff $\{true\}$ skip $\{Q\}$
- S. Cook: They are complete relative to being able to draw from the set of valid assertions about arithmetic.

Stephen Cook. Soundness and Completeness of an Axiom System for Program Verification. Siam Journal of Computing 1978

What can we do about pointers?

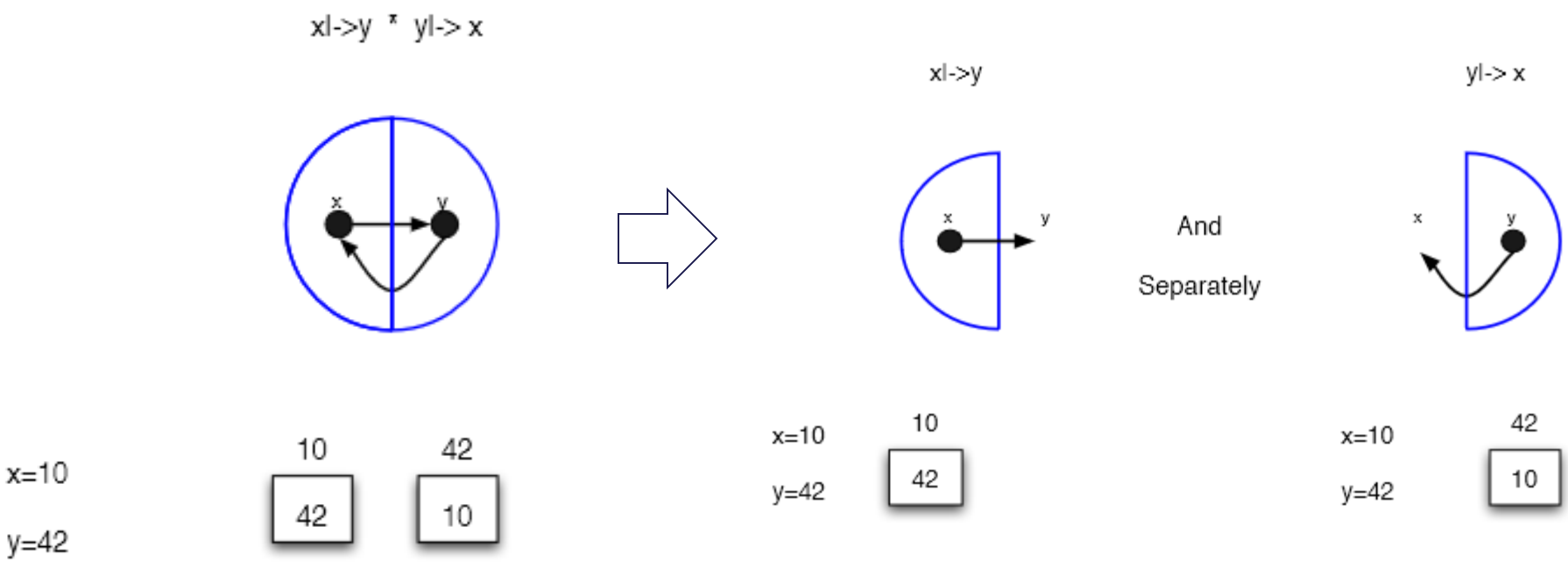
- Data structures are often disjointed (e.g., lists, sets...)
- Want to say which data can be accessed from the pointer, but also which data cannot.
- Let us define the language, semantics and logic extensions
  - Heap: map from location to value
  - Extend state: was just stack before, now it also has a heap: $(C, s_1,h_1) \Downarrow s_2,h_2$
  - Similar for a predicate satisfaction $s_1,h_1 \models P$

Presentation Based on:

- "Local Reasoning about Programs that Alter Data Structures" O'Hearn, Reynolds, Yang, CSL 2001
- "Separation Logic" O'Hearn CACM 2019
- MIT's Fundamentals of Program Analysis Course (by A. Solar-Lezama), Fall 2015 MIT OCW (CC BY-NC-SA)

Key Idea: Local Reasoning

- O'Hearn: "To understand how a program works, it should be possible for reasoning and specification to be confined to the cells that the program actually accesses. The value of any other cell will automatically remain unchanged."



Language with pointers

- Heap: $h \in H : \overbrace{\text{NaturalNumber}}^{\text{Location}} \rightarrow \overbrace{\text{Integer}}^{\text{Value}}$
- New statements that operate on heap
  - x := cons (e1, … en) : allocates a continuous segment of k cells
  - dispose (e) : deallocates the cell at address e
  - x := [e]: reads from the heap location pointed to by e
  - [e] := x: writes to the heap location pointed to by e

Language with pointers

- Heap: $h \in H : \overbrace{\text{NaturalNumber}}^{\text{Location}} \rightarrow \overbrace{\text{Integer}}^{\text{Value}}$
- New statements that operate on heap
  - x := cons (e1, … en) : allocates a continuous segment of k cells
  - dispose (e) : deallocates the cell at address e
  - x := [e]: reads from the heap location pointed to by e
  - [e] := x: writes to the heap location pointed to by e
- Example program:

```
z := cons (42, 0)
y := cons (123, z)
x := cons (555, y)
c := [ [x+1] ]
[x] := 999;
```

Language with pointers

- Heap: $h \in H : \overbrace{\text{NaturalNumber}}^{\text{Location}} \rightarrow \overbrace{\text{Integer}}^{\text{Value}}$
- New statements that operate on heap
  - x := cons (e1, … en) : allocates a continuous segment of k cells
  - dispose (e) : deallocates the cell at address e
  - x := [e]: reads from the heap location pointed to by e
  - [e] := x: writes to the heap location pointed to by e
- Extend the logic with the "spatial connectives":
  - emp: empty heap
  - $x \rightarrow v$: the stack variable x points to the value v on heap
  - P * Q: spatial conjunction – P and Q are separate parts of data structure (or structures): ***"and separately"***
    - Ensures that variables in P and Q are not aliases
  - P --* Q: spatial implication

Examples

- $(5 \rightarrow 3) * (6 \rightarrow 10)$ describes two adjacent cells whose contents are 3 and 10.

- $(E \rightarrow 3) \Rightarrow E \geq 0$

- $(E \rightarrow 3) * (F \rightarrow 3) \Rightarrow E \mathrel{!=} F$

Semantics of the Predicates

$$s, h \models B \quad \text{iff } [\![B]\!]s = true$$

$$s, h \models E \mapsto F \text{ iff } \{[\![E]\!]s\} = dom(h) \text{ and } h([\![E]\!]s) = [\![F]\!]s$$

$$s, h \models \texttt{false} \quad \text{never}$$

$$s, h \models P \Rightarrow Q \text{ iff if } s, h \models P \text{ then } s, h \models Q$$

$$s, h \models \forall x.P \quad \text{iff } \forall v \in \texttt{Ints}.\, [s \mid x \mapsto v], h \models P$$

$$s, h \models \texttt{emp} \quad \text{iff } h = [\,] \text{ is the empty heap}$$

h0#h1: h0 and h1 are disjoint

h0*h1: union of disjoint heaps

$$s, h \models P * Q \text{ iff } \exists h_0, h_1.\, h_0 \# h_1,\ h_0 * h_1 = h,\ s, h_0 \models P \text{ and } s, h_1 \models Q$$

$$s, h \models P \mathrel{-\!\!*} Q \text{ iff } \forall h'.\, \text{if } h' \# h \text{ and } s, h' \models P \text{ then } s, h * h' \models Q$$

From “Local Reasoning about Programs that Alter Data Structures” O’Hearn, Reynolds, Yang 2001

What can we do about pointers?

- So far programs had simple state (only stack

$$\{P\}\ C\ \{Q\}$$

$$\Leftrightarrow$$

$$\forall s_1 \forall s_2 .\ \ (\ s_1 \models P \wedge (C,s_1) \Downarrow s_2\ )\ \Rightarrow\ s_2 \models Q$$

- We can now extend the state with a heap h

$$\{P\}\ C\ \{Q\}$$

$$\Leftrightarrow$$

$$\forall s_1 \forall h_1 \forall s_2 \forall h_2 .\ \ (s_1,h_1 \models P \wedge (C,\ s_1,h_1) \Downarrow s_2,h_2\ )\ \Rightarrow s_2,h_2 \models Q$$

Some Additional Syntax

It will be convenient to have syntactic sugar for describing adjacent cells, and for an exact form of equality. We also have sugar for when E is an active address.

$$
\begin{array}{lcll}
E \mapsto F_0, ..., F_n & \triangleq & (E \mapsto F_0) * \cdots * (E + n \mapsto F_n) & \\
E \doteq F & \triangleq & (E = F) \wedge \texttt{emp} & \\
E \mapsto - & \triangleq & \exists y. E \mapsto y & (y \notin \mathrm{Free}(E))
\end{array}
$$

A characteristic property of $\doteq$ is the way it interacts with $*$:

$$(E \doteq F) * P \quad \Leftrightarrow \quad (E = F) \wedge P.$$

Example

- Formula: $(x \rightarrow a, o) * (x + o \rightarrow b, -o)$
- When $x = 17$, $o = 25$ the formula is true for which heap?

17	a
18	25

42	b
43	-25

Proof Rules

THE SMALL AXIOMS

$$\{E \mapsto -\}\ [E] := F\ \{E \mapsto F\}$$

$$\{E \mapsto -\}\ \texttt{dispose}(E)\ \{\texttt{emp}\}$$

$$\{x \doteq m\} x := \texttt{cons}(E_1, ..., E_k) \{x \mapsto E_1[m/x], ..., E_k[m/x]\}$$

$$\{x \doteq n\}\ x := E\ \{x \doteq (E[n/x])\}$$

$$\{E \mapsto n \wedge x = m\}\ x := [E]\ \{x = n \wedge E[m/x] \mapsto n\}$$

Proof Rules: Frame Rule

Contains the <u>store</u> variables that are being modifier

Modifies (x = E) = {x}

Modifies (x = [y]) = {x}

but

Modifies ([x] = y) = {}

// bc [x] is not a store var but an unnamed heap location!

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \text{Modifies}(C) \cap \text{Free}(R) = \{\}$$

Recall, semantically:

$$P * R \;\; \{ (s, h * h') \mid s,h \models P \;\wedge s,h' \models R \wedge h\#h' \}$$

Additional Proof Rules

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \text{Modifies}(C) \cap \text{Free}(R) = \{\}$$

Auxiliary Variable Elimination

$$\frac{\{P\}\,C\,\{Q\}}{\{\exists x.P\}\,C\,\{\exists x.Q\}} \quad x \notin \text{Free}(C)$$

Variable Substitution

$$\frac{\{P\}\,C\,\{Q\}}{(\{P\}\,C\,\{Q\})[E_1/x_1, ..., E_k/x_k]} \quad \begin{array}{l} \{x_1, ..., x_k\} \supseteq \text{Free}(P, C, Q), \text{ and} \\ x_i \in \text{Modifies}(C) \text{ implies} \\ E_i \text{ is a variable not free in any other } E_j \end{array}$$

Rule of Consequence

$$\frac{P' \Rightarrow P \qquad \{P\}\,C\,\{Q\} \qquad Q \Rightarrow Q'}{\{P'\}\,C\,\{Q'\}}$$

Example #1

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \ \text{Modifies}(C) \cap \text{Free}(R) = \{\}$$

- { (x→a, c) } [x] := b { (x→b, c) }

$$\frac{\dfrac{\{x \mapsto a\}\,[x] := b\,\{x \mapsto b\}}{\{(x \mapsto a) * (x+1 \mapsto c)\}\,[x] := b\,\{(x \mapsto b) * (x+1 \mapsto c)\}}\ \text{Frame}}{\{x \mapsto a, c\}\,[x] := b\,\{x \mapsto b, c\}}\ \text{Syntactic Sugar}$$

The overlap of free variables between $x + 1 \mapsto c$ and $[x] := b$ is allowed here because $\text{Modifies}([x] := b) = \{\}$.

Example: Circular list

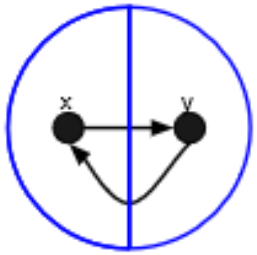
$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \ \text{Modifies}(C) \cap \text{Free}(R) = \{\}$$

{ (x → 0) * (y → 0) }

[x] = y;

[y] = x;

{ (x → y) * (y → x) }



Data structures: List

- Let list(a,b) describe an acyclic linked list segment from a to b.
  - $\text{list}(a,b) \Leftrightarrow (a=b \Rightarrow \text{emp}) \wedge$
    $(a!=b \Rightarrow \exists c\,.\,(a \rightarrow c) * \text{list}(c, b))$
  - $\text{list}'(a) \Leftrightarrow \text{list}(a, 0)$

- $\text{list}(x, t) * (t \rightarrow y) * \text{list}(y, 0)$
- We described three segments of the list…

Example: List

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \ \text{Modifies}(C) \cap \text{Free}(R) = \{\}$$

{ (x → z) * list(z) * (y → -) }

[x] = y;

{ (x → y) * list(z) * (y → -) }

[y] = z;

{ (x → y) * list(z) * (y → z) }

Data Structures: Trees

For our purposes a tree will either be an atom a or a pair (τ_1, τ_2) of trees. Here is an inductive definition of a predicate $\texttt{tree}\,\tau\,i$ which says when a number i represents a tree τ.

$$\begin{aligned}
\texttt{tree}\,a\,i &\overset{\Delta}{\iff} i = a \wedge \texttt{isatom?}(a) \wedge \texttt{emp} \\
\texttt{tree}\,(\tau_1, \tau_2)\,i &\overset{\Delta}{\iff} \exists x, y.\,(i \mapsto x, y) * (\texttt{tree}\,\tau_1\,x * \texttt{tree}\,\tau_2\,y)
\end{aligned}$$

The specification of the CopyTree procedure is

$$\{\texttt{tree}\,\tau\,p\}\ \texttt{CopyTree}(p; q)\ \{(\texttt{tree}\,\tau\,p) * (\texttt{tree}\,\tau\,q)\}.$$

Reasoning about Non-determinism

- Consider a non-deterministic condition:

$$C1 \; [] \; C2$$

- We execute one of C1 or C2

- Same as `if ( * ) then C1 else C2`
- This leads us to the new rule:

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{P\}\ C_2\ \{Q\}}{\{P\}\ C_1\ \mathbf{[]}\ C_2\ \{Q\}}$$

Exercise

- How do we write the rule for nondeterministic assignment?

X = E1 [] E2

From Non-determinism to Probability

- We execute the statement C1 with probability p and C2 with probability 1-p.

$$C1 \ [p] \ C2$$

- Commonly

$$X = E1 \ [p] \ X = E2$$

- We can think of the semantics of probabilistic choice as flipping a coin:

```
if ( flip(p) ) then C1 else C2
```

From Non-determinism to Probability

- Logic needs to be extended with probabilistic predicates
- Rules extend the ones from the Hoare Logic

$\{ \Pr(x = 1) > 3/4 \}$

$\quad y = x + \text{flip}(0.5) + \text{flip}(0.5)$

$\{ \Pr (y = 2) > 1/4 \}$

Reasoning about Termination

- Most rules are as before (but note the square brackets!)

$$\frac{}{[P\,[e/x]\,]\ x := e\ [P]}$$

$$\frac{[P]\ C_1\ [Q] \qquad [Q]\ C_2\ [R]}{[P]\ C_1;\ C_2\ [R]}$$

$$\frac{P \Rightarrow P' \qquad [P]\ C\ [Q'] \qquad Q' \Rightarrow Q}{[P]\ C\ [Q]}$$

$$\frac{[P \wedge B]\ C_1\ [Q] \qquad [P \wedge (\text{not } B)]\ C_2\ [Q]}{[P]\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{fi}\ [Q]}$$

Reasoning about Termination

- But loops are different

- Old (partial correctness):

$$\frac{\{ P \wedge B \}\ C\ \{ P \}}{\{ P \}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ \{ P \}}$$

- New?

Reasoning about Termination

- But loops are different

- Old (partial correctness):

$$\frac{\{ P \wedge B \}\ C\ \{ P \}}{\{ P \}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ \{ P \}}$$

- New?
- Intuition: imagine we have a counter; if we can prove the counter stops, we prove termination

Reasoning about Termination

$$\frac{\{P \wedge B\}\ C\ \{P\} \qquad \{P \wedge B \wedge t = z\}\ C\ \{t < z\} \qquad P \Rightarrow t \geq 0}{[\,P\,]\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ [\,P\,]}$$

- t is an integer expression (called bound or rank function)
- z is a fresh integer variable that does not appear anywhere in P,B,C

- z holds an initial value of t (since it is not changed by s)
- t is decreased with each iteration (2nd premise) and t is nonnegative if another iteration is possible (3rd premise).
- Thus no infinite computation is possible.

Example

if x is nonnegative and y is a positive integer, then S terminates with quo being the integer quotient and rem being the remainder of x divided by y

```
quo := 0;

rem := x;

while rem ≥ y do

    rem := rem – y;

    quo := quo + 1

od.
```

Example

if x is nonnegative and y is a positive integer, then S terminates with quo being the integer quotient and rem being the remainder of x divided by y

$\{x \geq 0 \wedge y > 0\}$

```
quo := 0;

rem := x;

while rem ≥ y do

    rem := rem - y;

    quo := quo + 1

od
```

$\{quo \cdot y + rem = x \wedge 0 \leq rem < y\}$

Example: Invariant?

$\{x \geq 0 \wedge y > 0\}$

```
quo := 0;

rem := x;

while_{Inv,t} rem ≥ y do

    rem := rem - y;

    quo := quo + 1

od
```

$\{quo \cdot y + rem = x \wedge 0 \leq rem < y\}$

$$\mathbf{Inv} \cong \mathbf{quo \cdot y + rem = x \wedge rem \geq 0 \wedge y > 0}$$

$$\mathbf{t} \cong \mathbf{rem}$$

$$\frac{\{P \wedge B\}\ C\ \{P\} \quad \{P \wedge B \wedge t = z\}\ C\ \{t < z\} \quad P \Rightarrow t \geq 0}{[P]\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ [P]}$$

Focus on the loop (1) (invariant - old)

{ quo · y + rem = x ∧ rem ≥ 0 ∧ y > 0 ∧ rem ≥ y }

```
rem := rem - y;

quo := quo + 1
```

{ quo · y + rem = x ∧ rem ≥ 0 ∧ y > 0 }

$$\frac{\{P \wedge B\}\ C\ \{P\} \qquad \{P \wedge B \wedge t = z\}\ C\ \{t < z\} \qquad P \Rightarrow t \geq 0}{[\,P\,]\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ [\,P\,]}$$

Focus on the loop (2-3) (termination)

$\{ quo \cdot y + rem = x \wedge rem \geq 0 \wedge y > 0 \wedge rem \geq y \wedge \mathbf{rem=z} \}$

```
rem := rem - y;

quo := quo + 1
```

$\{ \mathbf{rem < z} \}$

And finaly show P => rem ≥ 0 :

$(quo \cdot y + rem = x \wedge rem \geq 0 \wedge y > 0) \Rightarrow rem \geq 0$

$$\frac{\{P \wedge B\}\ C\ \{P\} \qquad \{P \wedge B \wedge t = z\}\ C\ \{t < z\} \qquad P \Rightarrow t \geq 0}{[P]\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ [P]}$$

Decomposition

- To prove [P] C [Q], we first establish its partial correctness, i.e. {P} C {Q}.
- Then, to show termination it suffices to prove the simpler formula [P] C [true]

- Then we have:

$$\frac{\{P\}\ C\ \{Q\} \qquad [P]\ C\ [true]}{[P]\ C\ [Q]}$$

Example: As Before for Partial Correctness

$\{x \geq 0 \wedge y > 0\}$

```
quo := 0;

rem := x;

while_Inv rem ≥ y do

    rem := rem - y;

    quo := quo + 1

od
```

$\{quo \cdot y + rem = x \wedge 0 \leq rem < y\}$

$\mathbf{Inv} \cong \quad \mathbf{quo \cdot y + rem = x}$
$\mathbf{\wedge\ rem \geq 0 \wedge y > 0}$

Example: Simpler for Total Correctness!

$\{x \geq 0 \wedge y > 0\}$

```
quo := 0;

rem := x;

while_{Inv,t} rem ≥ y do

    rem := rem - y;

    quo := quo + 1

od
```

$\{true\}$

$Inv \cong rem \geq 0 \wedge y > 0$

$t \cong rem$

Step 1 is simpler to prove than on the prev. slide

$$\frac{\{P \wedge B\}\ C\ \{P\} \qquad \{P \wedge B \wedge t = z\}\ C\ \{t < z\} \qquad P \Rightarrow t \geq 0}{[P]\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ [P]}$$

Focus on the loop (2-3) (termination; simpler)

{rem ≥ 0 ∧ y > 0 ∧ rem ≥ y ∧ **rem=z** }

```
rem := rem - y;

quo := quo + 1
```

{ **rem < z** }

And finally show P => rem ≥ 0 :

(rem ≥ 0 ∧ y > 0) => rem ≥ 0

$$\frac{\{P \wedge B\}\ C\ \{P\} \qquad \{P \wedge B \wedge t = z\}\ C\ \{t < z\} \qquad P \Rightarrow t \geq 0}{[\,P\,]\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ [\,P\,]}$$

Another Example

```
[ x = x0 ∧ y = y0 ∧ x > 0 ∧ x > y ]

tmp = x - y;



while tmp > 0 do

   x = x - 1;
   y = y + 1;
   tmp = tmp - 1;

od


[ x0>y0 => y=x0 ∧ x=y0 ]
```

$$\frac{\{P \wedge B\}\ C\ \{P\} \qquad \{P \wedge B \wedge t = z\}\ C\ \{t < z\} \qquad P \Rightarrow t \geq 0}{[P]\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{od}\ [P]}$$

Probabilistic Termination

```
heads := true;
tails := false;
x := flip(0.5);
while x = heads do
     x := flip(0.5)
od
```

- Does it always terminate?
- What is the probability of terminating?

Probabilistic Termination

```
heads := true;
tails := false;
x := flip(0.5);
while x = heads do
     x := flip(0.5)
od
```

- Does it always terminate? No
- What is the probability of terminating? 1.0
- The distinction can be important in practice

Alternatively Reason about Termination

- Prove Partial Correctness {P} while B do C {Q}
- Then prove the loop terminates using an orthogonal technique
- If all loops in the program terminate (and the control flow is structured) then the program will terminate too

Examples in Research

- Terminator (SAS'05, PLDI'06)
- Liveness Analysis (POPL'07)
- Proving non-termination (POPL'08)