CS228 : Logic for Computer Science

Module 4: Hoare Logic and Program Verification

Lecture 22: Total Correctness and Wrapping up

Instructor: S. Akshay

IIT Bombay, India

Logistics

Quiz 2

- **Tomorrow, Friday, Oct 23rd, 6pm**
- Venue:
  - CC 101: Roll No. 901–950
  - CC 103: Roll No. 951–1050
  - CC 105: Roll No. 1051–1100
  - PwD students who wish to avail compensatory time: KR 125
- No cheatsheets allowed

Syllabus

- Module 3 (Predicate Logic/FOL): all of Lectures 12–18
- Module 4 (Hoare logic): Lectures 19–21, up to partial correctness and loop invariants; *variants and total correctness are not included*

Recap: A proof calculus for program correctness

$$\frac{\{\varphi\}\ C_1\ \{\eta\} \qquad \{\eta\}\ C_2\ \{\psi\}}{\{\varphi\}\ C_1; C_2\ \{\psi\}}\ \text{Seq. Comp.} \qquad \frac{\{\varphi_1\}\ C_1\ \{\psi\} \qquad \{\varphi_2\}\ C_2\ \{\psi\}}{\{(B \to \varphi_1) \wedge (\neg B \to \varphi_2)\}\ \textbf{if}\ B\ \{C_1\}\ \textbf{else}\ \{C_2\}\ \{\psi\}}\ \text{ITE}$$

Recap: A proof calculus for program correctness

$$\frac{\{\varphi\}\ C_1\ \{\eta\} \qquad \{\eta\}\ C_2\ \{\psi\}}{\{\varphi\}\ C_1; C_2\ \{\psi\}}\ \text{Seq. Comp.}$$

$$\frac{\{\varphi_1\}\ C_1\ \{\psi\} \qquad \{\varphi_2\}\ C_2\ \{\psi\}}{\{(B \to \varphi_1) \wedge (\neg B \to \varphi_2)\}\ \textbf{if}\ B\ \{C_1\}\ \textbf{else}\ \{C_2\}\ \{\psi\}}\ \text{ITE}$$

$$\frac{}{\{\psi[E/x]\}\ x := E\ \{\psi\}}\ \text{Assignment}$$

$$\frac{\vdash_{AR} \varphi \to \varphi' \qquad \{\varphi'\}\ C\ \{\psi'\} \qquad \vdash_{AR} \psi' \to \psi}{\{\varphi\}\ C\ \{\psi\}}\ \text{Implied}$$

Recap: A proof calculus for program correctness

$$\frac{\{\varphi\}\ C_1\ \{\eta\} \qquad \{\eta\}\ C_2\ \{\psi\}}{\{\varphi\}\ C_1; C_2\ \{\psi\}}\ \text{Seq. Comp.}$$

$$\frac{\{\varphi_1\}\ C_1\ \{\psi\} \qquad \{\varphi_2\}\ C_2\ \{\psi\}}{\{(B \to \varphi_1) \wedge (\neg B \to \varphi_2)\}\ \textbf{if}\ B\ \{C_1\}\ \textbf{else}\ \{C_2\}\ \{\psi\}}\ \text{ITE}$$

$$\frac{}{\{\psi[E/x]\}\ x := E\ \{\psi\}}\ \text{Assignment}$$

$$\frac{\vdash_{AR} \varphi \to \varphi' \qquad \{\varphi'\}\ C\ \{\psi'\} \qquad \vdash_{AR} \psi' \to \psi}{\{\varphi\}\ C\ \{\psi\}}\ \text{Implied}$$

$$\frac{\{\eta \wedge B\}\ C\ \{\eta\}}{\{\eta\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Partial-while}$$

Recap: A proof calculus for program correctness

$$\frac{\{\varphi\}\ C_1\ \{\eta\} \qquad \{\eta\}\ C_2\ \{\psi\}}{\{\varphi\}\ C_1; C_2\ \{\psi\}}\ \text{Seq. Comp.}$$

$$\frac{\{\varphi_1\}\ C_1\ \{\psi\} \qquad \{\varphi_2\}\ C_2\ \{\psi\}}{\{(B \to \varphi_1) \wedge (\neg B \to \varphi_2)\}\ \textbf{if}\ B\ \{C_1\}\ \textbf{else}\ \{C_2\}\ \{\psi\}}\ \text{ITE}$$

$$\frac{}{\{\psi[E/x]\}\ x := E\ \{\psi\}}\ \text{Assignment}$$

$$\frac{\vdash_{AR} \varphi \to \varphi' \qquad \{\varphi'\}\ C\ \{\psi'\} \qquad \vdash_{AR} \psi' \to \psi}{\{\varphi\}\ C\ \{\psi\}}\ \text{Implied}$$

$$\frac{\{\eta \wedge B\}\ C\ \{\eta\}}{\{\eta\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Partial-while}$$

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

Recap: A proof calculus for program correctness

$$\frac{\{\varphi\}\ C_1\ \{\eta\} \qquad \{\eta\}\ C_2\ \{\psi\}}{\{\varphi\}\ C_1; C_2\ \{\psi\}}\ \text{Seq. Comp.}$$

$$\frac{\{\varphi_1\}\ C_1\ \{\psi\} \qquad \{\varphi_2\}\ C_2\ \{\psi\}}{\{(B \to \varphi_1) \wedge (\neg B \to \varphi_2)\}\ \textbf{if}\ B\ \{C_1\}\ \textbf{else}\ \{C_2\}\ \{\psi\}}\ \text{ITE}$$

$$\frac{}{\{\psi[E/x]\}\ x := E\ \{\psi\}}\ \text{Assignment}$$

$$\frac{\vdash_{AR} \varphi \to \varphi' \qquad \{\varphi'\}\ C\ \{\psi'\} \qquad \vdash_{AR} \psi' \to \psi}{\{\varphi\}\ C\ \{\psi\}}\ \text{Implied}$$

$$\frac{\{\eta \wedge B\}\ C\ \{\eta\}}{\{\eta\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Partial-while}$$

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

$\vdash_{par} \{\varphi\}\ P\ \{\psi\}$: derivable with Partial-while; $\vdash_{tot} \{\varphi\}\ P\ \{\psi\}$: with Total-while instead

These proof systems are sound: $\vdash_{par}$ implies $\models_{par}$, and $\vdash_{tot}$ implies $\models_{tot}$.

Recall: Partial-while on $\mathbf{Fac}_1$

$\{y = z!\}$

while $(z \neq x)$ $\{$

$\{y = z! \wedge z \neq x\}$	Inv Hyp. and Guard
$\{y \cdot (z+1) = (z+1)!\}$	Implied
$z := z + 1$	
$\{y \cdot z = z!\}$	Assignment
$y := y * z$	
$\{y = z!\}$	Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$

Recall: Partial-while on $\mathbf{Fac}_1$

$\{y = z!\}$

while $(z \neq x)$ $\{$

$\{y = z! \wedge z \neq x\}$	Inv Hyp. and Guard
$\{y \cdot (z+1) = (z+1)!\}$	Implied
$z := z + 1$	
$\{y \cdot z = z!\}$	Assignment
$y := y * z$	
$\{y = z!\}$	Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$

$$\frac{}{\{\psi[E/x]\}\ x := E\ \{\psi\}}$$

- Assignment: an axiom (nothing above the line). ψ is $y = z!$; replacing the assigned y by $y * z$ gives $y \cdot z = z!$

Recall: Partial-while on $\mathbf{Fac}_1$

$\{y = z!\}$

while $(z \neq x)$ $\{$

$\{y = z! \wedge z \neq x\}$	Inv Hyp. and Guard
$\{y \cdot (z+1) = (z+1)!\}$	Implied
$z := z + 1$	
$\{y \cdot z = z!\}$	Assignment
$y := y * z$	
$\{y = z!\}$	Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$

$$\frac{\boxed{\{\eta \wedge B\}\ C\ \{\eta\}}}{\{\eta\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}$$

- Premise = the body: η is $y = z!$, B is $(z \neq x)$, C is $z := z + 1;\ y := y * z$

Recall: Partial-while on $\mathbf{Fac}_1$

$\{y = z!\}$

while $(z \neq x)$ $\{$

$\{y = z! \wedge z \neq x\}$	Inv Hyp. and Guard
$\{y \cdot (z+1) = (z+1)!\}$	Implied
$z := z + 1$	
$\{y \cdot z = z!\}$	Assignment
$y := y * z$	
$\{y = z!\}$	Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$ Partial-while

$$\frac{\{\eta \wedge B\}\ C\ \{\eta\}}{\{\eta\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}$$

- Premise = the body: η is $y = z!$, B is $(z \neq x)$, C is $z := z + 1;\ y := y * z$
- Conclusion = the whole loop

Topic 22.1

Total Correctness

From partial to total correctness of while loops

For while loops

Total correctness = Partial correctness + Variant

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

From partial to total correctness of while loops

For while loops

Total correctness = Partial correctness + Variant

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

- Variant E is the integer expression
  - whose value is equal to the logical variable E_0 before the loop body is executed
  - and after one execution of the loop body, the value is strictly less than E_0 but still non-negative.

From partial to total correctness of while loops

For while loops

Total correctness = Partial correctness + Variant

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

- Variant E is the integer expression
  - whose value is equal to the logical variable E_0 before the loop body is executed
  - and after one execution of the loop body, the value is strictly less than E_0 but still non-negative.
- η is an invariant as before

From partial to total correctness of while loops

For while loops

Total correctness = Partial correctness + Variant

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

- Variant E is the integer expression
  - whose value is equal to the logical variable E_0 before the loop body is executed
  - and after one execution of the loop body, the value is strictly less than E_0 but still non-negative.
- η is an invariant as before
- For **Fac**$_1$: $E = x - z$

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$z := z + 1;$
out y	$y := y * z;$
	$\}$

From partial to total correctness of while loops

For while loops

Total correctness = Partial correctness + Variant

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

- Variant E is the integer expression
  - whose value is equal to the logical variable E_0 before the loop body is executed
  - and after one execution of the loop body, the value is strictly less than E_0 but still non-negative.
- η is an invariant as before
- For $\textbf{Fac}_1$: $E = x - z$
- **Quick recall:** the variant must decrease in every iteration. Why must it also stay non-negative?

$\textbf{Fac}_1$ *inp* x *out* y	$y := 1;$ $z := 0;$ **while** $(z \neq x)\{$ $\quad z := z + 1;$ $\quad y := y * z;$ $\}$

From partial to total correctness of while loops

For while loops

Total correctness = Partial correctness + Variant

$$\frac{\{\eta \wedge B \wedge 0 \leq E = E_0\}\ C\ \{\eta \wedge 0 \leq E < E_0\}}{\{\eta \wedge 0 \leq E\}\ \textbf{while}\ B\ \{C\}\ \{\eta \wedge \neg B\}}\ \text{Total-while}$$

- Variant E is the integer expression
  - whose value is equal to the logical variable E_0 before the loop body is executed
  - and after one execution of the loop body, the value is strictly less than E_0 but still non-negative.
- η is an invariant as before
- For $\textbf{Fac}_1$: $E = x - z$
- **Quick recall:** the variant must decrease in every iteration. Why must it also stay non-negative?
- Otherwise it could decrease forever and the loop need not terminate.

$\textbf{Fac}_1$ *inp* x *out* y	$y := 1;$ $z := 0;$ **while** $(z \neq x)\{$ $\quad z := z + 1;$ $\quad y := y * z;$ $\}$

Finally: Total correctness for the factorial example

Fac$_1$ *inp* x *out* y	$y := 1;$ $z := 0;$ **while** $(z \neq x)\{$ $\quad z := z + 1;$ $\quad y := y * z;$ $\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

$y := 1;$

$z := 0;$

while $(z \neq x)\{$

$\quad z := z + 1$

$\quad y := y * z$

$\}$

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
$\mathbf{Fac}_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}\ \mathbf{Fac}_1\ \{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$y := 1;$

$z := 0;$

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\quad z := z + 1$

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$

$\}$

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
$\mathbf{Fac}_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}\ \mathbf{Fac}_1\ \{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$y := 1;$

$z := 0;$

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
$\mathbf{Fac}_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}\ \mathbf{Fac}_1\ \{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$y := 1;$

$z := 0;$

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z+1) = (z+1)! \wedge 0 \leq x - (z+1) < E_0\}$

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$y := 1;$

$z := 0;$

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z+1) = (z+1)! \wedge 0 \leq x - (z+1) < E_0\}$ Implied **check!**

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$y := 1;$

$z := 0;$

$\{y = z! \wedge (0 \leq x - z)\}$

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z+1) = (z+1)! \wedge 0 \leq x - (z+1) < E_0\}$ Implied **check!**

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$ Total-while

This is where the guard is used.

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$\{x \geq 0\}$

$y := 1;$

$z := 0;$

$\{y = z! \wedge (0 \leq x - z)\}$

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z+1) = (z+1)! \wedge 0 \leq x - (z+1) < E_0\}$ Implied

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$ Total-while

$\{y = x!\}$

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$\{x \geq 0\}$

$y := 1;$

$z := 0;$

$\{y = z! \wedge (0 \leq x - z)\}$

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z+1) = (z+1)! \wedge 0 \leq x - (z+1) < E_0\}$ Implied

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$ Total-while

$\{y = x!\}$ Implied

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$\{x \geq 0\}$

$y := 1;$

$\{y = 0! \wedge 0 \leq x - 0\}$

$z := 0;$

$\{y = z! \wedge (0 \leq x - z)\}$ Assignment

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z + 1) = (z + 1)! \wedge 0 \leq x - (z + 1) < E_0\}$ Implied

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$ Total-while

$\{y = x!\}$ Implied

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$\{x \geq 0\}$

$\{1 = 0! \wedge 0 \leq x - 0\}$

$y := 1;$

$\{y = 0! \wedge 0 \leq x - 0\}$ Assignment

$z := 0;$

$\{y = z! \wedge (0 \leq x - z)\}$ Assignment

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z + 1) = (z + 1)! \wedge 0 \leq x - (z + 1) < E_0\}$ Implied

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$ Total-while

$\{y = x!\}$ Implied

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

$\{x \geq 0\}$

$\{1 = 0! \wedge 0 \leq x - 0\}$ Implied

$y := 1;$

$\{y = 0! \wedge 0 \leq x - 0\}$ Assignment

$z := 0;$

$\{y = z! \wedge (0 \leq x - z)\}$ Assignment

while $(z \neq x)\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z + 1) = (z + 1)! \wedge 0 \leq x - (z + 1) < E_0\}$ Implied

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

$\{y = z! \wedge \neg(z \neq x)\}$ Total-while

$\{y = x!\}$ Implied

Finally: Total correctness for the factorial example

	$y := 1;$
	$z := 0;$
Fac$_1$	**while** $(z \neq x)\{$
inp x	$\quad z := z + 1;$
out y	$\quad y := y * z;$
	$\}$

Show that

$\vdash_{tot} \{x \geq 0\}$ **Fac**$_1$ $\{y = x!\}$

Step 1: We guess

- Invariant: $y = z!$
- Variant $x - z$

(1) entry, (2) exit, (3) body keeps η and decreases E

$\{x \geq 0\}$

$\{1 = 0! \wedge 0 \leq x - 0\}$ Implied

$y := 1;$

$\{y = 0! \wedge 0 \leq x - 0\}$ Assignment

$z := 0;$

(1)$\{y = z! \wedge (0 \leq x - z)\}$ Assignment

while $(z \neq x)\{$

(3)$\{y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0\}$ Inv Hyp. and Guard

$\{y \cdot (z + 1) = (z + 1)! \wedge 0 \leq x - (z + 1) < E_0\}$ Implied

$\quad z := z + 1$

$\{y \cdot z = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\quad y := y * z$

$\{y = z! \wedge 0 \leq x - z < E_0\}$ Assignment

$\}$

(2)$\{y = z! \wedge \neg(z \neq x)\}$ Total-while

$\{y = x!\}$ Implied

Some exercises

Fac$_2$ *inp* x *out* y	$y := 1;$ **while** $(x \neq 0)\{$ $\quad y := y * x;$ $\quad x := x - 1;$ $\}$

Mult *inp* x, y *out* z	$a := 0;$ $z := 0;$ **while** $(a \neq y)\{$ $\quad z := z + x;$ $\quad a := a + 1;$ $\}$

Unk$_2$ *inp* x, y *out* d	$r := x;$ $d := 0;$ **while** $(r \geq y)\{$ $\quad r := r - y;$ $\quad d := d + 1;$ $\}$

Exercise 22.1

1. What is a variant for **Fac**$_2$?
2. Show that
   $\vdash_{tot} \{x = x_0 \wedge x \geq 0\}$ **Fac**$_2$ $\{y = x_0!\}$

Some exercises

Fac$_2$ *inp* x *out* y	$y := 1;$ **while** $(x \neq 0)\{$ $\quad y := y * x;$ $\quad x := x - 1;$ $\}$

Mult *inp* x, y *out* z	$a := 0;$ $z := 0;$ **while** $(a \neq y)\{$ $\quad z := z + x;$ $\quad a := a + 1;$ $\}$

Unk$_2$ *inp* x, y *out* d	$r := x;$ $d := 0;$ **while** $(r \geq y)\{$ $\quad r := r - y;$ $\quad d := d + 1;$ $\}$

Exercise 22.1

1. What is a variant for **Fac**$_2$?
2. Show that
   $\vdash_{tot} \{x = x_0 \wedge x \geq 0\}$ **Fac**$_2$ $\{y = x_0!\}$

Exercise 22.2

1. What is a variant for **Mult**?
2. Show that
   $\vdash_{tot} \{y \geq 0\}$ **Mult** $\{(z = x \cdot y)\}$

Some exercises

Fac$_2$ *inp* x *out* y	$y := 1;$ **while** $(x \neq 0)\{$ $\quad y := y * x;$ $\quad x := x - 1;$ $\}$

Mult *inp* x, y *out* z	$a := 0;$ $z := 0;$ **while** $(a \neq y)\{$ $\quad z := z + x;$ $\quad a := a + 1;$ $\}$

Unk$_2$ *inp* x, y *out* d	$r := x;$ $d := 0;$ **while** $(r \geq y)\{$ $\quad r := r - y;$ $\quad d := d + 1;$ $\}$

Exercise 22.1

1. What is a variant for **Fac**$_2$?
2. Show that
   $\vdash_{tot} \{x = x_0 \wedge x \geq 0\}$ **Fac**$_2$ $\{y = x_0!\}$

Exercise 22.2

1. What is a variant for **Mult**?
2. Show that
   $\vdash_{tot} \{y \geq 0\}$ **Mult** $\{(z = x \cdot y)\}$

Exercise 22.3

1. What is a variant for **Unk**$_2$?
2. Is $\vdash_{tot} \{y \neq 0\}$ **Unk**$_2$ $\{(x = d \cdot y + r) \wedge (r < y)\}$? If yes, prove it, else correct it.

Finding a variant can be hard: Collatz

- $x = 5$: $c = 5, 16, 8, 4, 2, 1$

Collatz
inp x
	$c := x$;
	while $(c \neq 1)\{$
	  if $(c \bmod 2 = 0)$ $c := c/2$;
	  else $c := 3 * c + 1$;
	$\}$

Finding a variant can be hard: Collatz

Collatz
inp x
	$c := x$;
	while $(c \neq 1)\{$
	  if $(c \bmod 2 = 0)$ $c := c/2$;
	  else $c := 3 * c + 1$;
	$\}$

- $x = 5$: $c = 5, 16, 8, 4, 2, 1$
- $x = 172$: 32 iterations; $x = 123456789$: 177 iterations

Finding a variant can be hard: Collatz

Collatz *inp* x	$c := x$;
	while $(c \neq 1)\{$
	  if $(c \bmod 2 = 0)$ $c := c/2$;
	  else $c := 3 * c + 1$;
	$\}$

- $x = 5$: $c = 5, 16, 8, 4, 2, 1$
- $x = 172$: 32 iterations; $x = 123456789$: 177 iterations
- **Quick question:** is c a variant? Is there one?

Finding a variant can be hard: Collatz

Collatz *inp* x	$c := x$;
	while $(c \neq 1)\{$
	  if $(c \bmod 2 = 0)$ $c := c/2$;
	  else $c := 3 * c + 1$;
	$\}$

- $x = 5$: $c = 5, 16, 8, 4, 2, 1$
- $x = 172$: 32 iterations; $x = 123456789$: 177 iterations
- **Quick question:** is c a variant? Is there one? c is not: it grows when c is odd!

Finding a variant can be hard: Collatz

Collatz *inp* x	$c := x$;
	while $(c \neq 1)\{$
	  if $(c \bmod 2 = 0)$ $c := c/2$;
	  else $c := 3 * c + 1$;
	$\}$

- $x = 5$: $c = 5, 16, 8, 4, 2, 1$
- $x = 172$: 32 iterations; $x = 123456789$: 177 iterations
- **Quick question:** is c a variant? Is there one? c is not: it grows when c is odd!
- Nobody knows whether it always terminates (the Collatz conjecture)

Finding a variant can be hard: Collatz

Collatz *inp* x	$c := x$;
	while $(c \neq 1)\{$
	  if $(c \bmod 2 = 0)$ $c := c/2$;
	  else $c := 3 * c + 1$;
	$\}$

- $x = 5$: $c = 5, 16, 8, 4, 2, 1$
- $x = 172$: 32 iterations; $x = 123456789$: 177 iterations
- **Quick question:** is c a variant? Is there one? c is not: it grows when c is odd!
- Nobody knows whether it always terminates (the Collatz conjecture)
- **Therefore** nobody knows whether a *variant* exists for this program either!

A few points to note and conclude

We "guessed" the invariant and variant!

- too weak *or* too strong: the proof fails. No algorithm finds the right invariant or variant in general (undecidable)!
- humans, search or an LLM can guess; a solver checks

A few points to note and conclude

We "guessed" the invariant and variant!

- too weak *or* too strong: the proof fails. No algorithm finds the right invariant or variant in general (undecidable)!
- humans, search or an LLM can guess; a solver checks

Annotations help programmers

- programmer gives the "intended" invariant / variant
- checking is largely automated by SMT solvers, e.g. Z3; the other steps are mechanical

A few points to note and conclude

We "guessed" the invariant and variant!

- too weak *or* too strong: the proof fails. No algorithm finds the right invariant or variant in general (undecidable)!
- humans, search or an LLM can guess; a solver checks

Annotations help programmers

- programmer gives the "intended" invariant / variant
- checking is largely automated by SMT solvers, e.g. Z3; the other steps are mechanical

Tools infer midconditions

- KeY-Hoare, Dafny, Why3, Frama-C/WP: weakest preconditions + SMT solver
- from annotations or from code; code alone is not always enough!

A few points to note and conclude

We "guessed" the invariant and variant!

- too weak *or* too strong: the proof fails. No algorithm finds the right invariant or variant in general (undecidable)!
- humans, search or an LLM can guess; a solver checks

Annotations help programmers

- programmer gives the "intended" invariant / variant
- checking is largely automated by SMT solvers, e.g. Z3; the other steps are mechanical

Tools infer midconditions

- KeY-Hoare, Dafny, Why3, Frama-C/WP: weakest preconditions + SMT solver
- from annotations or from code; code alone is not always enough!

Beyond: heaps, dynamic data, pointers

- *Separation logic* (Reynolds, O'Hearn)
- Meta's Infer is built on it

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and semantics
2. Natural deduction
3. Normal Forms

Applications: logic puzzles, checking arguments

Module 2: The Problem of Satisfiability

1. Propositional Resolution
2. DPLL and CDCL
3. SAT solver tool demo

Applications: HW/SW verification, planning

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and semantics
2. Natural deduction
3. Normal Forms

Applications: logic puzzles, checking arguments

Module 2: The Problem of Satisfiability

1. Propositional Resolution
2. DPLL and CDCL
3. SAT solver tool demo

Applications: HW/SW verification, planning

Module 3: Predicate Logic

1. Syntax and semantics of predicate logic
2. Natural deduction for FOL
3. Structures, power and limits of FOL
4. Normal forms and DPLL(T)

Applications: databases, SMT solvers

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and semantics
2. Natural deduction
3. Normal Forms

Applications: logic puzzles, checking arguments

Module 2: The Problem of Satisfiability

1. Propositional Resolution
2. DPLL and CDCL
3. SAT solver tool demo

Applications: HW/SW verification, planning

Module 3: Predicate Logic

1. Syntax and semantics of predicate logic
2. Natural deduction for FOL
3. Structures, power and limits of FOL
4. Normal forms and DPLL(T)

Applications: databases, SMT solvers

Module 4: Hoare Logic and Program Verification

1. Hoare triples; partial vs. total correctness
2. Proof rules for partial correctness
3. Loop invariants
4. Variants and total correctness

Applications: verifying code

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and semantics
2. Natural deduction
3. Normal Forms

Applications: logic puzzles, checking arguments

Module 2: The Problem of Satisfiability

1. Propositional Resolution
2. DPLL and CDCL
3. SAT solver tool demo

Applications: HW/SW verification, planning

Module 3: Predicate Logic

1. Syntax and semantics of predicate logic
2. Natural deduction for FOL
3. Structures, power and limits of FOL
4. Normal forms and DPLL(T)

Applications: databases, SMT solvers

Module 4: Hoare Logic and Program Verification

1. Hoare triples; partial vs. total correctness
2. Proof rules for partial correctness
3. Loop invariants
4. Variants and total correctness

Applications: verifying code

Next: Module 5, Temporal Logic — LTL, CTL and model checking

Topic 22.2

Module 5: Temporal Logic

The story so far

Propositional logic



a *static snapshot*:
which facts are true?

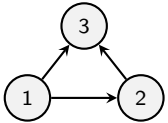
The story so far

Propositional logic



a *static snapshot*:
which facts are true?

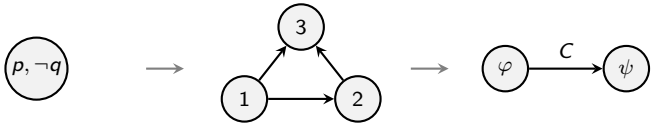
Predicate logic



a *richer* snapshot:
objects $1, 2, 3$ and
the relation $<$;
formulas like
$\forall x \exists y \;\; x < y$

The story so far

Propositional logic → **Predicate logic** → **Hoare logic**



a *static snapshot*: which facts are true?

a *richer* snapshot: objects $1, 2, 3$ and the relation $<$; formulas like $\forall x \exists y \;\; x < y$

two snapshots: before $\rightarrow$ after, then it halts

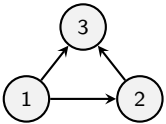
The story so far

Propositional logic



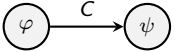
a *static snapshot*:
which facts are true?

Predicate logic



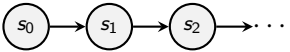
a *richer* snapshot:
objects $1, 2, 3$ and
the relation $<$;
formulas like
$\forall x \exists y \;\; x < y$

Hoare logic



two snapshots:
before $\rightarrow$ after,
then it halts

Temporal logic



a whole *film*:
infinitely many
snapshots!

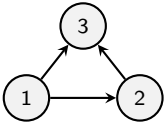
The story so far

Propositional logic



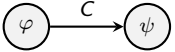
a *static snapshot*:
which facts are true?

Predicate logic



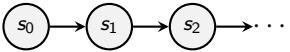
a *richer* snapshot:
objects $1, 2, 3$ and
the relation $<$;
formulas like
$\forall x \exists y \;\; x < y$

Hoare logic



two snapshots:
before $\rightarrow$ after,
then it halts

Temporal logic



a whole *film*:
infinitely many
snapshots!

But why would a program need infinitely many snapshots?

Beyond programs that stop

- Hoare logic: a program starts, *halts*, and we compare before and after

Beyond programs that stop

- Hoare logic: a program starts, *halts*, and we compare before and after
- But an operating system, a lift controller, a pacemaker should *never* stop!

Beyond programs that stop

- Hoare logic: a program starts, *halts*, and we compare before and after
- But an operating system, a lift controller, a pacemaker should *never* stop!
- We want to say: "every request is eventually granted", "the lift never moves with its doors open"

Beyond programs that stop

- Hoare logic: a program starts, *halts*, and we compare before and after
- But an operating system, a lift controller, a pacemaker should *never* stop!
- We want to say: "every request is eventually granted", "the lift never moves with its doors open"
- **Quick question:** can we write these as Hoare triples?

Beyond programs that stop

- Hoare logic: a program starts, *halts*, and we compare before and after
- But an operating system, a lift controller, a pacemaker should *never* stop!
- We want to say: "every request is eventually granted", "the lift never moves with its doors open"
- **Quick question:** can we write these as Hoare triples? Not really! These programs have no "end"!

Beyond programs that stop

- Hoare logic: a program starts, *halts*, and we compare before and after
- But an operating system, a lift controller, a pacemaker should *never* stop!
- We want to say: "every request is eventually granted", "the lift never moves with its doors open"
- **Quick question:** can we write these as Hoare triples? Not really! These programs have no "end"!

Amir Pnueli, 1977

- introduced temporal logic to reason about programs ("The temporal logic of programs", FOCS 1977)
- Turing Award, 1996, "for seminal work introducing temporal logic into computing science"



Amir Pnueli

Temporal Logics

In propositional and predicate (first-order) logic

- Models or assignments were static
- A formula is either true or false in a given model
- So we could for e.g., say $m \models \varphi$.

Temporal Logics

In propositional and predicate (first-order) logic

- Models or assignments were static
- A formula is either true or false in a given model
- So we could for e.g., say $m \models \varphi$.

Idea of Temporal logics

- What if models are richer and dynamic.
- i.e., Models have states and transitions.
- A formula's truth may change depending on the state of a given model!
- So we could for e.g., say $M, s \models \varphi$.

Temporal Logics

In propositional and predicate (first-order) logic

- Models or assignments were static
- A formula is either true or false in a given model
- So we could for e.g., say $m \models \varphi$.

Idea of Temporal logics

- What if models are richer and dynamic.
- i.e., Models have states and transitions.
- A formula's truth may change depending on the state of a given model!
- So we could for e.g., say $M, s \models \varphi$.

Quick recall: what was a *model* in propositional logic? In first-order logic?

Temporal Logics

In propositional and predicate (first-order) logic

- Models or assignments were static
- A formula is either true or false in a given model
- So we could for e.g., say $m \models \varphi$.

Idea of Temporal logics

- What if models are richer and dynamic.
- i.e., Models have states and transitions.
- A formula's truth may change depending on the state of a given model!
- So we could for e.g., say $M, s \models \varphi$.

Quick recall: what was a *model* in propositional logic? In first-order logic? A valuation; a structure. Now: a graph of states and transitions!

Notions of Time

Temporal logics are dynamic

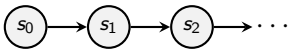
- Truth of a formula depends on which state within the model you are in currently.
- So you have to talk about how the formula evolves over time.

Notions of Time

Temporal logics are dynamic

- Truth of a formula depends on which state within the model you are in currently.
- So you have to talk about how the formula evolves over time.

Linear-time $\rightarrow$ LTL



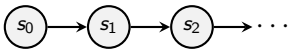
view time as a path (or a set of paths):
one future at each moment

Notions of Time

Temporal logics are dynamic

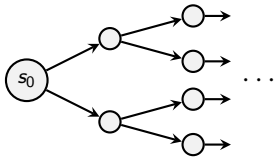
- Truth of a formula depends on which state within the model you are in currently.
- So you have to talk about how the formula evolves over time.

Linear-time → LTL



view time as a path (or a set of paths):
one future at each moment

Branching-time → CTL



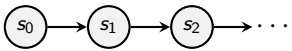
view time as a tree, rooted at present:
many possible futures

Notions of Time

Temporal logics are dynamic

- Truth of a formula depends on which state within the model you are in currently.
- So you have to talk about how the formula evolves over time.

Linear-time → LTL

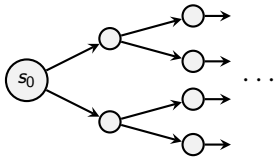


view time as a path (or a set of paths):
one future at each moment



Loki: the Sacred Timeline,
the TVA prunes every branch

Branching-time → CTL



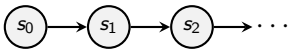
view time as a tree, rooted at present:
many possible futures

Notions of Time

Temporal logics are dynamic

- Truth of a formula depends on which state within the model you are in currently.
- So you have to talk about how the formula evolves over time.

Linear-time → LTL

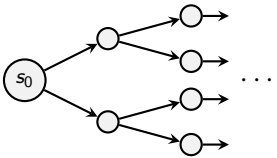


view time as a path (or a set of paths):
one future at each moment



Loki: the Sacred Timeline,
the TVA prunes every branch

Branching-time → CTL



view time as a tree, rooted at present:
many possible futures



Doctor Strange: millions of futures,
only one where we win

Time can be modelled in other ways too

Not the only way to distinguish:

- You can also differentiate discrete and continuous time

Time can be modelled in other ways too

Not the only way to distinguish:

- You can also differentiate discrete and continuous time
- There are logics for each one!

Time can be modelled in other ways too

Not the only way to distinguish:

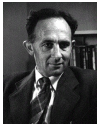
- You can also differentiate discrete and continuous time
- There are logics for each one!
- But we will focus on two of them: LTL and CTL.

Time can be modelled in other ways too

Not the only way to distinguish:

- You can also differentiate discrete and continuous time
- There are logics for each one!
- But we will focus on two of them: LTL and CTL.

A little history



Arthur Prior
late 1950s:
tense logic



Amir Pnueli
1977: temporal logic
for programs



Edmund Clarke
1981: CTL



E. Allen Emerson
1981: CTL



Joseph Sifakis
early 1980s:
model checking

Clarke, Emerson and Sifakis shared the 2007 Turing Award for founding *model checking*.

Still relevant today? Model checking where failure is expensive

Still relevant today? Model checking where failure is expensive

- Space: NASA's *FRET*: structured English $\rightarrow$ temporal logic (e.g. grasping spent rocket stages)
- Blockchain: Ethereum's 3SF consensus checked with *TLA+* and Apalache (2025)
- Industry: *TLA+ Foundation* (2023: AWS, Oracle, Microsoft); AWS's *Kani* found Firecracker bugs

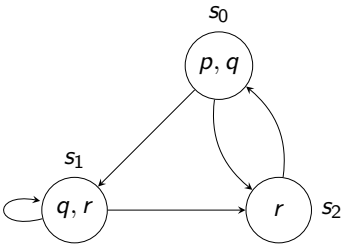
Still relevant today? Model checking where failure is expensive

- Space: NASA's *FRET*: structured English $\rightarrow$ temporal logic (e.g. grasping spent rocket stages)
- Blockchain: Ethereum's 3SF consensus checked with *TLA+* and Apalache (2025)
- Industry: *TLA+ Foundation* (2023: AWS, Oracle, Microsoft); AWS's *Kani* found Firecracker bugs

Specify in temporal logic, check by model checking: wherever failure is expensive!

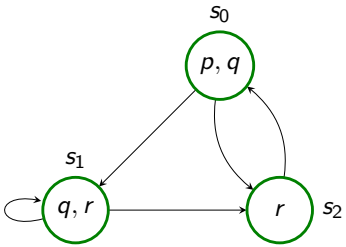
Transition systems as models

To reason about temporal properties of systems, we need a richer model...



A directed graph: each node (state) contains the atomic propositions true in that state

Transition systems as models

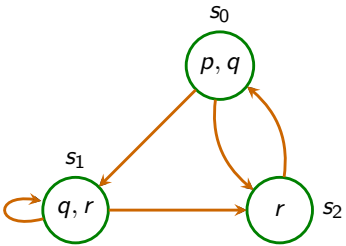


A directed graph: each node (state) contains the atomic propositions true in that state

A Transition system (also called a *Kripke structure*) is a triple $M = (S, \rightarrow, L)$ where,

- S is a set of states

Transition systems as models

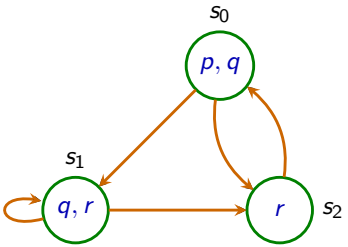


A directed graph: each node (state) contains the atomic propositions true in that state

A Transition system (also called a *Kripke structure*) is a triple $M = (S, \rightarrow, L)$ where,

- S is a set of states
- $\rightarrow$ is a binary relation on S called transition relation s.t., $\forall s \in S, \exists s' \in S$ with $s \rightarrow s'$.

Transition systems as models

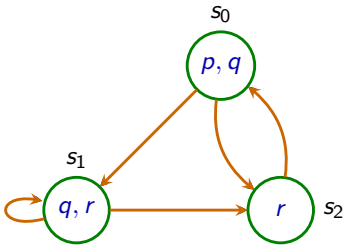


A directed graph: each node (state) contains the atomic propositions true in that state

A Transition system (also called a *Kripke structure*) is a triple $M = (S, \rightarrow, L)$ where,

- S is a set of states
- $\rightarrow$ is a binary relation on S called transition relation s.t., $\forall s \in S, \exists s' \in S$ with $s \rightarrow s'$.
- $L : S \rightarrow \mathcal{P}(AP)$ where
  - AP is a set of atomic propositions: statements about system, e.g., *Printer is busy*
  - 2^{AP} is its power set: the set of all subsets.

Transition systems as models



A directed graph: each node (state) contains the atomic propositions true in that state

L is like a valuation: in propositional logic, $m \subseteq AP$ assigned truth values to all propositions; now each state has its own $L(s) \subseteq AP$.

A Transition system (also called a *Kripke structure*) is a triple $M = (S, \rightarrow, L)$ where,

- S is a set of states
- $\rightarrow$ is a binary relation on S called transition relation s.t., $\forall s \in S, \exists s' \in S$ with $s \rightarrow s'$.
- $L : S \rightarrow \mathcal{P}(AP)$ where
  - AP is a set of atomic propositions: statements about system, e.g., *Printer is busy*
  - 2^{AP} is its power set: the set of all subsets.