Design and Analysis of Algorithms CS218M

Correctness of Algorithms

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

Programs and Assertions

Programs (e expressions, b boolean expr.)

```
x:=e
S1;S2
if b then S1 else S2 fi
while b do S od
```

A State assigns a value to each variable.

A program starts in an initial state. It ends in a final state or does not terminate.

Assertions

Assertions

Conditions on state. They specify a subset of states. E.g. $x > y$.
Formally, assertions are formulae of first-order logic.

Assertions use logical connectives.

$P \wedge Q$	P and Q
$P \vee Q$	P or Q
$\neg P$	not P
$P \Rightarrow Q$	whenever P is true so is Q

Reasoning

$AXIOMS \models P \Rightarrow Q$

A Simple Program

Problem

Compute quotient q and reminder r of integers x divided by y.

```
r:=x; q:=0;
while r > y do
        r:=r-y;  q:=q+1
od
```

A Simple Program

Problem

Compute quotient q and reminder r of integers x divided by y.

```
r:=x; q:=0;
while r > y do
            r:=r-y;  q:=q+1
od
```

x	y	q	r
8	3	2	2
8	0		
-8	3	0	-8
6	3	1	3

A Simple Program

Problem

Compute quotient q and reminder r of integers x divided by y.

$\{0 < y \quad \wedge \quad 0 \leq x\}$ Precondition

```
r:=x; q:=0;
while r > y do
            r:=r-y;  q:=q+1
od
```

x	y	q	r
8	3	2	2
8	0		
-8	3	0	-8
6	3	1	3

A Simple Program

Problem

Compute quotient q and reminder r of integers x divided by y.

$\{0 < y \quad \wedge \quad 0 \leq x\}$ Precondition

```
r:=x; q:=0;
while r > y do
        r:=r-y;  q:=q+1
od
```

$\{x = y * q + r \quad \wedge \quad 0 \leq r < y\}$ Postcondition

x	y	q	r
8	3	2	2
8	0		
-8	3	0	-8
6	3	1	3

Program specification

$\{\ P\ \}\quad S\quad \{\ Q\ \}$ Hoare Triple.

- S Program (fragment)
- P Precondition

  Assumed to be true when S starts.
- Q Postcondition

  Required to be true when S terminates.

Advantages

- Clear and Unambiguous articulation of what program must do.
- Separation of concern: User versus developer.

  interface specification.
- Can be formally verified.

Annotated Program

$\{0 < y \ \wedge \ 0 \leq x\}$ (1)

r:=x; q:=0; (2)

$\{0 < y \ \wedge \ 0 \leq x \ \wedge \ r = x \ \wedge \ q = 0\}$ (3)

$\{inv : 0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (4)

while ↑ r≥y do (6)

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \leq r\}$ (7)

r:=r-y; q:=q+1 (8)

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (9)

od (10)

$\{r < y \ \wedge \ 0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (11)

$\{x = y * q + r \ \wedge \ 0 \leq r < y\}$ (12)

Assertions

Pre-condition and post-condition.

Location Invariants

- Control location: a position before a program statement
- Location Invariant: Condition which is true every time control reaches the location.

Assertions

Pre-condition and post-condition.

Location Invariants

- Control location: a position before a program statement
- Location Invariant: Condition which is true every time control reaches the location.

Loop Invariant

Consider *while b do S od*.

- A condition which holds every time condition b is tested.

Given predicate Q
$Q[e/x]$ denotes Q with x substituted by e
E.g. $x < 0[x+1/x]$ gives $x + 1 < 0$.

Assignment

$$\{Q[e/x]\} \quad x := e \quad \{Q\}$$

R1

Example: $\{x + 1 < 0\}$ x:=x+1 $\{x < 0\}$

Sequential Composition

$$\frac{\{P\}\ S_1\ \{Q_1\}, \quad Q_1 \Rightarrow Q_2, \quad \{Q_2\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}}$$

R2

Hoare Logic (2)

Consequence

$$\frac{P \Rightarrow P_1, \quad \{P_1\}\ S\ \{Q_1\}, \quad Q_1 \Rightarrow Q}{\{P\}\ S\ \{Q\}}$$

R3

Conditional Statement

$$\frac{\{P \wedge b\}\ S_1\ \{Q\}, \quad \{P \wedge \neg b\}\ S_2\ \{Q\}}{\{P\}\ \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi }\ \{Q\}}$$

R4

Proofs using Hoare Logic Rules

Claim: $\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \leq r\}$

r:=r-y; q:=q+1

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$

Proofs using Hoare Logic Rules

Claim: $\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \leq r\}$

r:=r-y; q:=q+1

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$

Proof

(1)

(2)

(3)

(4)

(5)

(6)

Proofs using Hoare Logic Rules

Claim: $\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \leq r\}$

r:=r-y; q:=q+1

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$

Proof

(1)

(2)

(3)

(4)

q:=q+1 (5)

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (6)

Proofs using Hoare Logic Rules

Claim: $\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \leq r\}$

r:=r-y; q:=q+1

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$

Proof

(1)

(2)

(3)

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * (q + 1) + r\}$ (4)

q:=q+1 (5)

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (6)

Proofs using Hoare Logic Rules

Claim: $\{0 \le r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \le r\}$
r:=r-y; q:=q+1
$\{0 \le r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$

Proof

(1)

(2)

r:=r-y; (3)

$\{0 \le r \ \wedge \ 0 < y \ \wedge \ x = y * (q + 1) + r\}$ (4)

q:=q+1 (5)

$\{0 \le r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (6)

Proofs using Hoare Logic Rules

Claim: $\{0 \leq r \wedge 0 < y \wedge x = y * q + r \wedge y \leq r\}$
r:=r-y; q:=q+1
$\{0 \leq r \wedge 0 < y \wedge x = y * q + r\}$

Proof

(1)

$\{0 \leq r - y \wedge 0 < y \wedge x = y * (q + 1) + (r - y)\}$ (2)

r:=r-y; (3)

$\{0 \leq r \wedge 0 < y \wedge x = y * (q + 1) + r\}$ (4)

q:=q+1 (5)

$\{0 \leq r \wedge 0 < y \wedge x = y * q + r\}$ (6)

Proofs using Hoare Logic Rules

Claim: $\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \leq r\}$
r:=r-y; q:=q+1
$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$

Proof

$\{0 \leq r \ \wedge \ 0 < y \leq r \ \wedge \ x = y * q + r\}$ (1)

$\{0 \leq r - y \ \wedge \ 0 < y \ \wedge$ (2)
$x = y * (q + 1) + (r - y)\}$

r:=r-y; (3)

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * (q + 1) + r\}$ (4)

q:=q+1 (5)

$\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (6)

While Statement

Let P be loop invariant. It holds every time the loop condition is tested.

bound Fucl t

$$\frac{\{P \wedge b\}\ S\ \{P\}}{\{P\}\ \texttt{while}\ b\ \text{do}\ S\ \text{od}\ \ \{P \wedge \neg b\}}$$

Proving Termination

Let t be bound function. Bound function is integer valued total function.

While Rule

RS

$$\frac{\begin{array}{l} Q \ \Rightarrow\ P \\ \{P \wedge b\}\ S\ \{P\} \\ P \wedge \neg b \ \Rightarrow\ R \\ P \wedge b \ \Rightarrow\ t > 0 \\ \{P \wedge b \wedge t = k\}\ S\ \{t < k\} \end{array}}{\{Q\}\ \texttt{while}\ b\ \text{do}\ S\ \text{od}\ \ \{R\}}$$

initialzetion part

maintain

$t = 0 \Rightarrow \neg b$

Progress

While Rule Intuition

Premises:

Initially the invariant holds. (PR1)
Each loop iteration preserves loop invariant. (PR2)
Each loop iteration decrements bound function from a positive value. (PR4)
Loop terminates before making bound function non-positive. (PR5)

Conclusion:

On termination invariant must hold and also loop condition must be false. These together imply post condition by PR3.
The loop must terminate as bound function cannot decrement indefinitely from a positive value.

Annotated Program

$\{0 < y \ \wedge \ 0 \leq x\}$ (1)

r:=x; q:=0; (2)

$\{0 < y \ \wedge \ 0 \leq x \ \wedge \ r = x \ \wedge \ q = 0\}$ (3)

$\{inv : 0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (4)

$\{bound : \ r\}$ (5)

while $\uparrow$ r≥y do (6)

  $\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r \ \wedge \ y \leq r\}$ (7)

  r:=r-y; q:=q+1 (8)

  $\{0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (9)

od inv $\wedge$ (10)

$\{r < y \ \wedge \ 0 \leq r \ \wedge \ 0 < y \ \wedge \ x = y * q + r\}$ (11)

$\{x = y * q + r \ \wedge \ 0 \leq r < y\}$ (12)

$\{a, b : int,\ b \geq 0\}$ MULT $\{z = a * b\}$

strategy "repeated add a to z b times."

$z := 0;\ x := a;\ y := b;$ $O(m)$

$\{Inv:\ z + x * y = a * b \wedge y \geq 0\}$

while $(y > 0)$ $\{bound:\ y\}$ $O(m)$ $O(2^m)$

$\{Inv \wedge y > 0\}$

~~Body~~ $z := z + x;$ $O(m)$ $O(2^m)$

$y := y - 1$

$\{Inv\}$

do

$\{Inv \wedge y \leq 0\}$

$\{z = a * b\}$

$O(m * 2^m)$

m-bit numbers

Efficient Multiplication

```
{0 ≤ b}
x:=a; y:=b; z:=0 ;
{Inv: z + x*y = a*b ∧ y ≥ 0}
while y > 0 do
      {Inv ∧ y > 0}
      if even(y) then
            {Inv ∧ y > 0 ∧ even(y)}
            x:=x+x;  y:=y/2
            {Inv}
      else

            y:=y-1;  z:=z+x

      fi
      {Inv}
od

{z = a * b}
```

Efficient Multipliation (2)

```
{0 ≤ b}
x:=a; y:=b; z:=0 ;
{inv :   0 ≤ y   ∧ z + x ∗ y = a ∗ b}
{bound :   y}
while y > 0 do
        {inv  ∧  y > 0}
        if even(y) then
                {inv  ∧  y > 0  ∧  even(x)}
                x:=x+x;  y:=y/2
                {inv}
        else
                {inv  ∧  y > 0  ∧  ¬even(x)}
                y:=y-1;  z:=z+x
                {inv}
        fi
        {inv}
od
{inv  ∧  y ≤ 0}
{z = a ∗ b}
```

Cost   Count
$O(n)$ 1
$O(n)$ $O(n)$
$O(n)$ $O(n)$
$O(n \times n)$
$O(n^2)$

Founders of Formal Verification

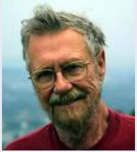
First Order Logic for Assertions

Alan Turing

Bob Floyd

Tony Hoare

Edsgar Dijkstra



Other Contributors

- O.J. Dahl (Data structuring)
- S. Cook (Relative Completeness)

David Gries, The Science of Programming, Springer-Verlag.

Essentials of First-Order Predicate Logic

A language for describing mathematical structures.

A **structure** $\mathcal{U} = (S, F, G)$

S - set of values.

called *Domain*, written as $|\mathcal{U}|$

F - set of functions over S

G - set of relations over S

Pair (F, G) is called the signature.

Examples

ω	Natural Numbers
$\Re$	Real Numbers
Bool	$(\{0, 1\},\ \ \{\wedge, \neg\},\ \ \{=\})$

state

$div(x+1, y)$

Some valid properties of ω

$$\forall y.\ (0 < y \ \vee\ 0 = y)$$
$$\forall x.\ x < x + 1$$
$$\forall x, y, z.\ (x * (y + z) \ = \ x * y + x * z)$$

$div(x, y)$ means x "divides" y

$$div(x, y) \ \stackrel{\text{def}}{=}\ \exists z.\ x * z = y$$

$div(3, 4)$

$\exists z.\ 3 * z = 4$

$prime(x)$ means x is a prime.

$$prime(x) \ \stackrel{\text{def}}{=}\ \forall y.$$
$$(div(y, x) \ \Rightarrow \ y = 1 \vee y = x)$$

$\omega, \sigma(x) = 4 \not\models prime(x)$

Natural Numbers ω

Domain $\{0, 1, 2, \ldots\}$
Functions 0, 1, $+$, $*$
Relations $<$, $=$

What do f.o.l. formulas over ω look like?

Terms

- Examples: x+0*y 1*z

Natural Numbers ω

Domain $\{0, 1, 2, \ldots\}$
Functions 0, 1, $+$, $*$
Relations $<$, $=$

What do f.o.l. formulas over ω look like?

Terms

- Examples: x+0*y 1*z
- Syntax: $t ::= x \mid f(t_1, \ldots, t_n)$

Natural Numbers ω

Domain $\{0, 1, 2, \ldots\}$
Functions 0, 1, $+$, $*$
Relations $<$, $=$

What do f.o.l. formulas over ω look like?

Terms

- Examples: x+0*y 1*z
- Syntax: $t \quad ::= \quad x \mid f(t_1, \ldots, t_n)$
- State (valuation) $\sigma : Var \rightarrow |\mathcal{U}|$.
  E.g. $\sigma(x) = 3, \sigma(y) = 4, \sigma(z) = 2$.

Natural Numbers ω

Domain $\{0, 1, 2, \ldots\}$
Functions 0, 1, $+$, $*$
Relations $<$, $=$

What do f.o.l. formulas over ω look like?

Terms

- Examples: x+0*y 1*z
- Syntax: $t \quad ::= \quad x \mid f(t_1, \ldots, t_n)$
- State (valuation) $\sigma : Var \rightarrow |\mathcal{U}|$.
  E.g. $\sigma(x) = 3, \sigma(y) = 4, \sigma(z) = 2$.
- Value of term t in structure $\mathcal{U}$ and state σ is $\hat{\sigma}(t) \in |\mathcal{U}|$

Natural Numbers ω

Domain $\{0, 1, 2, \ldots\}$
Functions 0, 1, $+$, $*$
Relations $<$, $=$

What do f.o.l. formulas over ω look like?

Terms

- Examples: x+0*y 1*z
- Syntax: $t \quad ::= \quad x \mid f(t_1, \ldots, t_n)$
- State (valuation) $\sigma : Var \rightarrow |\mathcal{U}|$.
  E.g. $\sigma(x) = 3, \sigma(y) = 4, \sigma(z) = 2$.
- Value of term t in structure $\mathcal{U}$ and state σ is $\hat{\sigma}(t) \in |\mathcal{U}|$
- $\hat{\sigma}(x + 0 * y) \ = \ 3 + 0 * 4 \ = \ 3.$

Natural Numbers ω

Domain $\{0, 1, 2, \ldots\}$
Functions 0, 1, +, $*$
Relations $<$, $=$

What do f.o.l. formulas over ω look like?

Terms

- Examples: x+0*y 1*z
- Syntax: $t \quad ::= \quad x \mid f(t_1, \ldots, t_n)$
- State (valuation) $\sigma : Var \rightarrow |\mathcal{U}|$.
  E.g. $\sigma(x) = 3, \sigma(y) = 4, \sigma(z) = 2$.
- Value of term t in structure $\mathcal{U}$ and state σ is $\hat{\sigma}(t) \in |\mathcal{U}|$
- $\hat{\sigma}(x + 0 * y) \;=\; 3 + 0 * 4 \;=\; 3.$
- Semantics: $\hat{\sigma}(x) \;=\; \sigma(x)$
  $\hat{\sigma}(f(t_1, \ldots, t_n)) \;=\; f(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n))$

First order logic (cont)

Atomic Formulae

- Example: $x + 0 * y \; < \; 1 * z$

First order logic (cont)

Atomic Formulae

- Example: $x + 0 * y \ < \ 1 * z$
- Syntax: $\psi \quad ::= \quad t_1 = t_2 \ \mid \ R(t_1, \ldots, t_n)$

First order logic (cont)

Atomic Formulae

- Example: $x + 0 * y \ < \ 1 * z$
- Syntax: $\psi \quad ::= \quad t_1 = t_2 \ \mid \ R(t_1, \ldots, t_n)$
- $\mathcal{U}, \sigma \models \psi$ denotes that ψ evaluates to true in $\mathcal{U}, \sigma$.
  $\mathcal{U}, \sigma \not\models \psi$ denotes that ψ evaluates to false in $\mathcal{U}, \sigma$.

First order logic (cont)

Atomic Formulae

- Example: $x + 0 * y \ < \ 1 * z$
- Syntax: $\psi \quad ::= \quad t_1 = t_2 \mid R(t_1, \ldots, t_n)$
- $\mathcal{U}, \sigma \models \psi$ denotes that ψ evaluates to true in $\mathcal{U}, \sigma$.
  $\mathcal{U}, \sigma \not\models \psi$ denotes that ψ evaluates to false in $\mathcal{U}, \sigma$.
- Let $\sigma(x) = 3, \sigma(y) = 4, \sigma(z) = 2$. Then,
  $\omega, \sigma \not\models (x + 0 * y \ < \ 1 * z)$. (why?)

First order logic (cont)

Atomic Formulae

- Example: $x + 0 * y \; < \; 1 * z$
- Syntax: $\psi \quad ::= \quad t_1 = t_2 \;\mid\; R(t_1, \ldots, t_n)$
- $\mathcal{U}, \sigma \models \psi$ denotes that ψ evaluates to true in $\mathcal{U}, \sigma$.
  $\mathcal{U}, \sigma \not\models \psi$ denotes that ψ evaluates to false in $\mathcal{U}, \sigma$.
- Let $\sigma(x) = 3, \sigma(y) = 4, \sigma(z) = 2$. Then,
  $\omega, \sigma \not\models (x + 0 * y \; < \; 1 * z)$. (why?)
- Semantics:

$$\mathcal{U}, \sigma \models t_1 = t_2 \quad \textbf{iff} \quad \hat{\sigma}(t_1) = \hat{\sigma}(t_2)$$

$$\mathcal{U}, \sigma \models R(t_1, \ldots, t_n) \quad \textbf{iff}$$

$$R(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n))$$

Formulas

- Formula ϕ is made of atomic formulas using boolean connectives $\wedge, \vee, \neg, \Rightarrow$ as well as quantifiers $\exists x.\phi$ and $\forall x.\phi$.

Formulas

- Formula ϕ is made of atomic formulas using boolean connectives $\wedge, \vee, \neg, \Rightarrow$ as well as quantifiers $\exists x.\phi$ and $\forall x.\phi$.
- E.g. $(\forall y.\ (x < y \vee x = y))$.

Formulas

- Formula ϕ is made of atomic formulas using boolean connectives $\wedge, \vee, \neg, \Rightarrow$ as well as quantifiers $\exists x.\phi$ and $\forall x.\phi$.
- E.g. $(\forall y.\ (x < y \vee x = y))$.
- Syntax: $\phi \quad ::= \quad \psi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \neg\phi \ \mid \ \exists x.\phi$.

Formulas

- Formula ϕ is made of atomic formulas using boolean connectives $\wedge, \vee, \neg, \Rightarrow$ as well as quantifiers $\exists x.\phi$ and $\forall x.\phi$.
- E.g. $(\forall y.\ (x < y \vee x = y))$.
- Syntax: $\phi \quad ::= \quad \psi \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ \neg\phi \ \mid \ \exists x.\phi$.
- $\mathcal{U}, \sigma \models \phi$ denotes that ϕ evaluates to true in $\mathcal{U}, \sigma$.

Formulas

- Formula ϕ is made of atomic formulas using boolean connectives $\wedge, \vee, \neg, \Rightarrow$ as well as quantifiers $\exists x.\phi$ and $\forall x.\phi$.
- E.g. $(\forall y.\ (x < y \vee x = y))$.
- Syntax: $\phi \quad ::= \quad \psi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists x.\phi$.
- $\mathcal{U}, \sigma \models \phi$ denotes that ϕ evaluates to true in $\mathcal{U}, \sigma$.
- Formula $\exists x.\phi$ states that there exists a choice of value of x (ignoring the value given by $\sigma(x)$) which makes ϕ true. Formula $\forall x.\phi$ states that all choice of value of x (ignoring the value given by $\sigma(x)$) make ϕ true.

Formulas

- Formula ϕ is made of atomic formulas using boolean connectives $\wedge, \vee, \neg, \Rightarrow$ as well as quantifiers $\exists x.\phi$ and $\forall x.\phi$.
- E.g. $(\forall y.\ (x < y \vee x = y))$.
- Syntax: $\phi \quad ::= \quad \psi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists x.\phi$.
- $\mathcal{U}, \sigma \models \phi$ denotes that ϕ evaluates to true in $\mathcal{U}, \sigma$.
- Formula $\exists x.\phi$ states that there exists a choice of value of x (ignoring the value given by $\sigma(x)$) which makes ϕ true. Formula $\forall x.\phi$ states that all choice of value of x (ignoring the value given by $\sigma(x)$) make ϕ true.
- Let $\sigma(x) = 0$. Then, $\omega, \sigma \not\models (\forall y.\ (x < y \vee x = y))$. (why?)

Formulas

- Formula ϕ is made of atomic formulas using boolean connectives $\wedge, \vee, \neg, \Rightarrow$ as well as quantifiers $\exists x.\phi$ and $\forall x.\phi$.
- E.g. $(\forall y.\ (x < y \vee x = y))$.
- Syntax: $\phi \quad ::= \quad \psi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists x.\phi$.
- $\mathcal{U}, \sigma \models \phi$ denotes that ϕ evaluates to true in $\mathcal{U}, \sigma$.
- Formula $\exists x.\phi$ states that there exists a choice of value of x (ignoring the value given by $\sigma(x)$) which makes ϕ true. Formula $\forall x.\phi$ states that all choice of value of x (ignoring the value given by $\sigma(x)$) make ϕ true.
- Let $\sigma(x) = 0$. Then, $\omega, \sigma \not\models (\forall y.\ (x < y \vee x = y))$. (why?)
- Semantics: σ' is x-variant of σ if $\sigma(y) = \sigma'(y)$ for all $y \neq x$.

$$\mathcal{U}, \sigma \models \exists x.\phi \quad \textbf{iff}$$

$$\mathcal{U}, \sigma' \models \phi \quad \text{for some } x\text{-variant } \sigma' \text{ of } \sigma$$

Quicksort

$$
\begin{array}{l}
Sorted(A,i,j) \stackrel{\text{def}}{=} \\
\qquad 1 \le i \le j \le n \quad \Rightarrow \\
\qquad\qquad \forall i'. i \le i' < j \Rightarrow A[i'] \le A[i'+1]
\end{array}
$$

$$
\begin{array}{l}
Partition(A,i,j,k) \stackrel{\text{def}}{=} \\
\qquad 1 \le i \le j \le k \le n \quad \wedge \\
\qquad (\forall i'.(i \le i' < j \Rightarrow A[i'] \le A[j]) \quad \wedge \\
\qquad (\forall k'.(j < k' \le k \Rightarrow A[j] \le A[k'])
\end{array}
$$

Then,

$$
\models \left(\begin{array}{ll} & Partition(A,i,j,k) \\ \wedge & Sorted(A,i,j-1) \\ \wedge & Sorted(A,j+1,k) \end{array} \right) \Rightarrow Sorted(A,i,k)
$$