

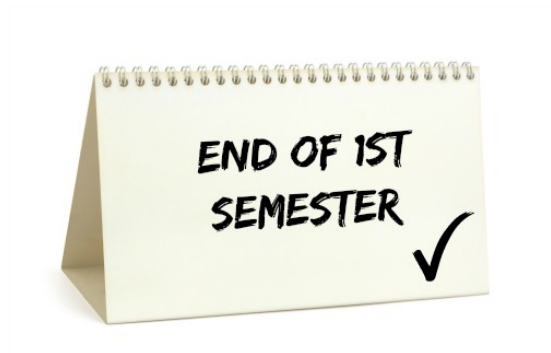
## 2-1 The Correctness of Algorithms

Hengfeng Wei

hfwei@nju.edu.cn

Feb. 27, 2020





Mathematical Logic

Set Theory

Abstract Algebra



Data Structures

Algorithms

# Design

Design

Analysis (Time & Space)

Design

Analysis (Time & Space)

**Correctness Proof**

Correctness proof of algorithms is very **important**.

## Algorithms to Live By



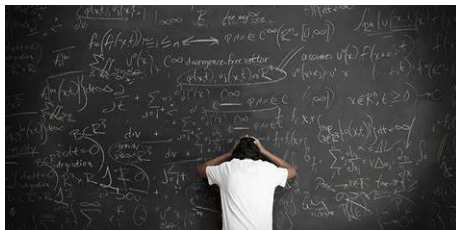
The  
COMPUTER SCIENCE  
of  
HUMAN DECISIONS

Brian Christian and Tom Griffiths



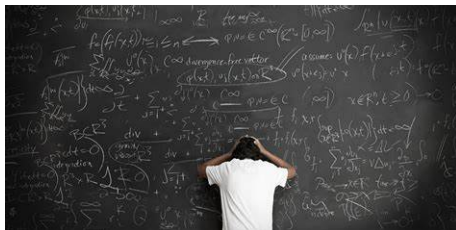
“We live in a world run by algorithms.”

Correctness proof of algorithms is very **hard**.





Correctness proof of algorithms is very **hard**.



Writing correctness proof needs understanding.

Understanding is hard.

We learn correctness proof of algorithms by:

We learn correctness proof of algorithms by:

- ▶ Learning general proof methods (Hoare Logic)
  - ▶ (Loop) Invariants for “partial correctness”
  - ▶ Variants for “termination”

We learn correctness proof of algorithms by:

- ▶ Learning general proof methods (Hoare Logic)
  - ▶ (Loop) Invariants for “partial correctness”
  - ▶ Variants for “termination”
- ▶ Learning by examples
  - ▶ REVERSE( $S$ )
  - ▶ DIVISION( $n, m$ )
  - ▶ EUCLID( $a, b$ )

We learn correctness proof of algorithms by:

- ▶ Learning general proof methods (Hoare Logic)
  - ▶ (Loop) Invariants for “partial correctness”
  - ▶ Variants for “termination”
- ▶ Learning by examples
  - ▶ REVERSE( $S$ )
  - ▶ DIVISION( $n, m$ )
  - ▶ EUCLID( $a, b$ )
- ▶ Learning by DIY (Doing It Yourself)

*What We Talk About*  
*When We Talk About **Correctness of Algorithms** ...*

*What We Talk About*  
*When We Talk About **Correctness of Algorithms** ...*

What is the correctness of algorithms?

## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.



## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.

Alg:  $B \leftarrow \text{Sort}(A)$

## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.

Alg:  $B \leftarrow \text{Sort}(A)$

Spec:  $B$  is sorted

## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.

Alg:  $B \leftarrow \text{Sort}(A)$

Spec:  $B$  is sorted

$$\forall 0 \leq i < n - 1 : B[i] \leq B[i + 1]$$

## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.

Alg:  $B \leftarrow \text{Sort}(A)$

Spec:  $B$  is sorted

Alg:  $d \leftarrow \text{Euclid}(a, b)$

$$\forall 0 \leq i < n - 1 : B[i] \leq B[i + 1]$$

## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.

Alg:  $B \leftarrow \text{Sort}(A)$

Spec:  $B$  is sorted

$$\forall 0 \leq i < n - 1 : B[i] \leq B[i + 1]$$

Alg:  $d \leftarrow \text{Euclid}(a, b)$

Spec :  $d = \text{gcd}(a, b)$

## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.

Alg:  $B \leftarrow \text{Sort}(A)$

Spec:  $B$  is sorted

$$\forall 0 \leq i < n - 1 : B[i] \leq B[i + 1]$$

Alg:  $d \leftarrow \text{Euclid}(a, b)$

Spec :  $d = \text{gcd}(a, b)$

We use **specification languages** to specify specs.

## Definition (Correctness of Algorithms (Not A Formal Definition))

An algorithm is **considered** correct if it meets its **specification**.

Alg:  $B \leftarrow \text{Sort}(A)$

Spec:  $B$  is sorted

$$\forall 0 \leq i < n - 1 : B[i] \leq B[i + 1]$$

Alg:  $d \leftarrow \text{Euclid}(a, b)$

Spec :  $d = \text{gcd}(a, b)$

We use **specification languages** to specify specs.

Generally, they are **mathematical logic + theory of domain knowledge**.

# Correctness: Partial Correctness & Total Correctness



## Correctness: Partial Correctness & Total Correctness

### Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

## Correctness: Partial Correctness & Total Correctness

### Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

### Definition (Partial Correctness)

*If* the algorithm terminates, it meets its specification.

## Correctness: Partial Correctness & Total Correctness

### Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

### Definition (Partial Correctness)

*If* the algorithm terminates, it meets its specification.

### Definition (Total Correctness (Revisited))

The algorithm indeed terminates **and** it meets its specification.

## Separating “partial correctness” from “termination”

Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

## Separating “partial correctness” from “termination”

### Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

- ▶ They are intrinsically different for serious theoretical reasons.

## Separating “partial correctness” from “termination”

### Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

- ▶ They are intrinsically different for serious theoretical reasons.
- ▶ Different **proof methods** for them (**Hoare Logic**)
  - ▶ (Loop) Invariants for “partial correctness”
  - ▶ Variants for “termination”

## Separating “partial correctness” from “termination”

### Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

- ▶ They are intrinsically different for serious theoretical reasons.
- ▶ Different **proof methods** for them (**Hoare Logic**)
  - ▶ (Loop) Invariants for “partial correctness”
  - ▶ Variants for “termination”
- ▶ “Termination” is often much easier for sequential algorithms.







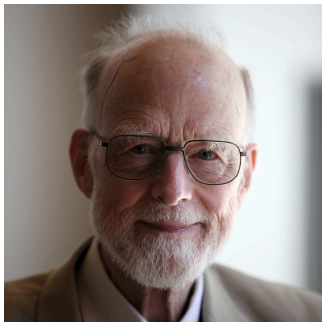
Robert W. Floyd (1936 ~ 2001)

Turing Award (1978)

## ASSIGNING MEANINGS TO PROGRAMS<sup>1</sup>

**Introduction.** This paper attempts to provide an adequate basis for formal definitions of the meanings of programs in appropriately defined programming languages, in such a way that a rigorous standard is established for proofs about computer programs, including proofs of correctness, equivalence, and termination. The basis of our approach is the notion of an interpretation of a program: that is, an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken. To prevent an interpretation from being chosen arbitrarily, a condition is imposed on each command of the program. This condition guarantees that whenever a command is reached by way of a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. Then by induction on the number of commands executed, one sees that if a program is entered by a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. By this means, we may prove certain properties of programs, particularly properties of the form: "If the initial values of the program variables satisfy the relation  $R_1$ , the final values on completion will satisfy the relation  $R_2$ ." Proofs of termination are dealt with by showing that each step of a program decreases some entity which cannot decrease indefinitely.

“Assigning Meanings to Programs”  
(1967)



Tony Hoare (1934 ~ )  
Turing Award (1980)

## An Axiomatic Basis for Computer Programming

C. A. R. HOARE

*The Queen's University of Belfast,\* Northern Ireland*

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

“An Axiomatic Basis for Computer  
Programming” (1969)

Hoare triple :  $\{P\} R \{Q\}$

Hoare triple :  $\{P\} R \{Q\}$

$P$  : Pre-condition     $R$  : Program     $Q$  : Post-condition

Hoare triple :  $\boxed{\{P\} R \{Q\}}$

$P$  : Pre-condition     $R$  : Program     $Q$  : Post-condition

*If the precondition  $P$  holds,  
the postcondition  $Q$  should also hold  
after executing the program  $R$ .*

Hoare triple :  $\boxed{\{P\} R \{Q\}}$

$P$  : Pre-condition     $R$  : Program     $Q$  : Post-condition

Partial Correctness:

*If the precondition  $P$  holds,  
the postcondition  $Q$  should also hold  
*after* executing the program  $R$ .*

Hoare triple :  $\{P\} R \{Q\}$

$P$  : Pre-condition     $R$  : Program     $Q$  : Post-condition

Partial Correctness:

*If the precondition  $P$  holds,  
the postcondition  $Q$  should also hold  
*after* executing the program  $R$ .*

Hoare logic provides the inference rules.

$$\boxed{\{P\} \ R \ \{Q\}}$$



$$\boxed{\{P\} R \{Q\}}$$

$R$  consists of

- ▶ “ $x \leftarrow a$ ”
- ▶ “ $S; T$ ”
- ▶ “if  $B$  then  $S$  else  $T$ ”
- ▶ “while  $B$  do  $S$ ”

$$\boxed{\{P\} R \{Q\}}$$

$$\{x = 42\} \ y \leftarrow x + 1 \ \{y = 43\}$$

$$\boxed{\{P\} R \{Q\}}$$

$$\{x = 42\} \ y \leftarrow x + 1 \ \{y = 43\}$$

$$\{x + 1 \leq N\} \ x \leftarrow x + 1 \ \{x \leq N\}$$

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{\{P\} S \{Q\} \quad , \quad \{Q\} T \{R\}}{\{P\} S; T \{R\}}$$

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{\{B \wedge P\} S \{Q\} \quad , \quad \{\neg B \wedge P\} T \{Q\}}{\{P\} \text{ if } B \text{ then } S \text{ else } T \{Q\}}$$

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{?}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{?}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$$\frac{P \implies I, \quad \{I \wedge B\} S \{I\}, \quad I \wedge \neg B \implies Q}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{?}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$I : \text{Loop Invariant}$



$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \textbf{ while } B \textbf{ do } S \{Q\}}$$

$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \textbf{ while } B \textbf{ do } S \{Q\}}$$

$Q$  : How to find  $I$ ?

$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \textbf{ while } B \textbf{ do } S \{Q\}}$$

$Q$  : How to find  $I$ ?

The general answer is “I don’t know.”



$$\boxed{\{P\} R \{Q\}}$$

$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$Q$  : How to show its termination?

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$Q$  : How to show its termination?

$$\frac{\{B \wedge t \in D \wedge t = z\} S \{t \in D \wedge t < z\}}{(\text{while } B \text{ do } S) \text{ will terminate}}$$

$(D, <)$  : a well-ordered set

$$\boxed{\{P\} R \{Q\}}$$

$$\frac{P \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P\} \textbf{ while } B \textbf{ do } S \{Q\}}$$

$Q$  : How to show its termination?

$$\frac{\{B \wedge t \in D \wedge t = z\} S \{t \in D \wedge t < z\}}{(\textbf{while } B \textbf{ do } S) \text{ will terminate}}$$

$(D, <)$  : a well-ordered set

$t$  : *Loop Variant*



$$\frac{\{B \wedge t \in D \wedge t = z\} S \{t \in D \wedge t < z\}}{(\textbf{while } B \textbf{ do } S) \text{ will terminate}}$$

$$\frac{\{B \wedge t \in D \wedge t = z\} S \{t \in D \wedge t < z\}}{(\textbf{while } B \textbf{ do } S) \text{ will terminate}}$$

*Q* : How to find *t*?

$$\frac{\{B \wedge t \in D \wedge t = z\} \ S \ \{t \in D \wedge t < z\}}{(\mathbf{while} \ B \ \mathbf{do} \ S) \text{ will terminate}}$$

*Q* : How to find *t*?

The general answer is “I don’t know.”

$$\frac{\{B \wedge t \in D \wedge t = z\} \ S \ \{t \in D \wedge t < z\}}{(\text{while } B \text{ do } S) \text{ will terminate}}$$

*Q* : How to find *t*?

The general answer is “I don’t know.”





- ▶  $\text{REVERSE}(X)$
- ▶  $\text{DIVISION}(n, m)$
- ▶  $\text{EUCLID}(a, b)$

---

```
1: procedure REVERSE( $S$ )
2:
3:    $X \leftarrow S$                                  $\triangleright$  keep  $S$  intact
4:    $Y \leftarrow \Lambda$                          $\triangleright$   $Y$  is to store the result
5:
6:   while  $X \neq \Lambda$  do
7:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
8:      $X \leftarrow \text{TAIL}(X)$ 
9:
10:  return  $Y$ 
```

---

---

```
1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$                                  $\triangleright$  keep  $S$  intact
4:    $Y \leftarrow \Lambda$                            $\triangleright$   $Y$  is to store the result
5:
6:   while  $X \neq \Lambda$  do
7:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
8:      $X \leftarrow \text{TAIL}(X)$ 
9:    $Q : Y = \text{reverse}(S)$ 
10:  return  $Y$ 
```

---

---

```
1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$                                  $\triangleright$  keep  $S$  intact
4:    $Y \leftarrow \Lambda$                          $\triangleright$   $Y$  is to store the result
5:    $I : S = \text{reverse}(Y) \cdot X$ 
6:   while  $X \neq \Lambda$  do
7:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
8:      $X \leftarrow \text{TAIL}(X)$ 
9:    $Q : Y = \text{reverse}(S)$ 
10:  return  $Y$ 
```

---



$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$\frac{P' \Rightarrow I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \Rightarrow Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$P' \Rightarrow I$$

$$\frac{P' \Rightarrow I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \Rightarrow Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$P' \Rightarrow I$$

$$X = \text{reverse}(\Lambda) \cdot X$$

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

 $\{I \wedge B\} S \{I\}$

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$\{I \wedge B\} S \{I\}$$

$$S = \text{reverse}(Y) \cdot X \wedge X \neq \Lambda \quad (1)$$

$$Y' \leftarrow \text{HEAD}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \text{TAIL}(X) \quad (3)$$

---


$$S = \text{reverse}(Y') \cdot X' \quad (4)$$

$$S = \text{reverse}(Y) \cdot X \wedge X \neq \Lambda \quad (1)$$

$$Y' \leftarrow \text{HEAD}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \text{TAIL}(X) \quad (3)$$

---


$$S = \text{reverse}(Y') \cdot X' \quad (4)$$

$$S = \text{reverse}(Y') \cdot X'$$

$$S = \text{reverse}(Y) \cdot X \wedge X \neq \Lambda \quad (1)$$

$$Y' \leftarrow \text{HEAD}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \text{TAIL}(X) \quad (3)$$

---


$$S = \text{reverse}(Y') \cdot X' \quad (4)$$

$$\begin{aligned} S &= \text{reverse}(Y') \cdot X' \\ &= \text{reverse}(\text{HEAD}(X) \cdot Y) \cdot \text{TAIL}(X) \end{aligned}$$



$$S = \text{reverse}(Y) \cdot X \wedge X \neq \Lambda \quad (1)$$

$$Y' \leftarrow \text{HEAD}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \text{TAIL}(X) \quad (3)$$

---


$$S = \text{reverse}(Y') \cdot X' \quad (4)$$

$$S = \text{reverse}(Y') \cdot X'$$

$$= \text{reverse}(\text{HEAD}(X) \cdot Y) \cdot \text{TAIL}(X)$$

$$= \text{reverse}(Y) \cdot \text{reverse}(\text{HEAD}(X)) \cdot \text{TAIL}(X)$$

$$S = \text{reverse}(Y) \cdot X \wedge X \neq \Lambda \quad (1)$$

$$Y' \leftarrow \text{HEAD}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \text{TAIL}(X) \quad (3)$$

---


$$S = \text{reverse}(Y') \cdot X' \quad (4)$$

$$S = \text{reverse}(Y') \cdot X'$$

$$= \text{reverse}(\text{HEAD}(X) \cdot Y) \cdot \text{TAIL}(X)$$

$$= \text{reverse}(Y) \cdot \text{reverse}(\text{HEAD}(X)) \cdot \text{TAIL}(X)$$

$$= \text{reverse}(Y) \cdot (\text{HEAD}(X) \cdot \text{TAIL}(X))$$

$$S = \text{reverse}(Y) \cdot X \wedge X \neq \Lambda \quad (1)$$

$$Y' \leftarrow \text{HEAD}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \text{TAIL}(X) \quad (3)$$

$$S = \text{reverse}(Y') \cdot X' \quad (4)$$

$$S = \text{reverse}(Y') \cdot X'$$

$$= \text{reverse}(\text{HEAD}(X) \cdot Y) \cdot \text{TAIL}(X)$$

$$= \text{reverse}(Y) \cdot \text{reverse}(\text{HEAD}(X)) \cdot \text{TAIL}(X)$$

$$= \text{reverse}(Y) \cdot (\text{HEAD}(X) \cdot \text{TAIL}(X))$$

$$= \text{reverse}(Y) \cdot X$$

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$I \wedge \neg B \implies Q$$

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

1: **procedure** REVERSE( $S$ )

2:      $P : \top$

3:      $X \leftarrow S$

4:      $Y \leftarrow \Lambda$

5:      $P' : X = S \wedge Y = \Lambda$

6:      $I : S = \text{reverse}(Y) \cdot X$

7:     **while**  $X \neq \Lambda$  **do**

8:          $Y \leftarrow \text{HEAD}(X) \cdot Y$

9:          $X \leftarrow \text{TAIL}(X)$

10:      $Q : Y = \text{reverse}(S)$

11:     **return**  $Y$

---

$$I \wedge \neg B \implies Q$$

$$S = \text{reverse}(Y) \cdot X \quad (1)$$

$$X = \Lambda \quad (2)$$

---


$$Y = \text{reverse}(S) \quad (3)$$

$$\frac{\{B \wedge t \in D \wedge t = z\} S \{t \in D \wedge t < z\}}{(\text{while } B \text{ do } S) \text{ will terminate}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$$\frac{\{B \wedge t \in D \wedge t = z\} S \{t \in D \wedge t < z\}}{(\text{while } B \text{ do } S) \text{ will terminate}}$$

---

```

1: procedure REVERSE( $S$ )
2:    $P : \top$ 
3:    $X \leftarrow S$ 
4:    $Y \leftarrow \Lambda$ 
5:    $P' : X = S \wedge Y = \Lambda$ 
6:    $I : S = \text{reverse}(Y) \cdot X$ 
7:   while  $X \neq \Lambda$  do
8:      $Y \leftarrow \text{HEAD}(X) \cdot Y$ 
9:      $X \leftarrow \text{TAIL}(X)$ 
10:   $Q : Y = \text{reverse}(S)$ 
11:  return  $Y$ 

```

---

$t = \text{length}(X)$



$$S = \text{reverse}(Y) \cdot X$$

$$S = \text{reverse}(Y) \cdot X$$

$$\text{TotalWork} = \text{WorkDone} \oplus \text{WorkToDo}$$

$$S = \text{reverse}(Y) \cdot X$$

$$\boxed{\text{TotalWork} = \text{WorkDone} \oplus \text{WorkToDo}}$$

$$\underbrace{\text{reverse}(S)}_{\text{TotalWork}} = \underbrace{\text{reverse}(X)}_{\text{WorkToDo}} \cdot \underbrace{Y}_{\text{WorkDone}}$$

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \textbf{ while } B \textbf{ do } S \{Q\}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q :$ 
11:  return ( $q, r$ )

```

---

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \textbf{ while } B \textbf{ do } S \{Q\}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$$P' \implies I$$

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \textbf{ while } B \textbf{ do } S \{Q\}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$$P' \implies I$$

$$n = m \cdot 0 + n$$



$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

---

<pre> 1: <b>procedure</b> DIVISION(<math>n, m</math>) 2:   <math>P : n &gt; 0 \wedge m &gt; 0</math> 3:   <math>r \leftarrow n</math> 4:   <math>q \leftarrow 0</math> 5:   <math>P' : r = n \wedge q = 0</math> 6:   <math>I : n = mq + r</math> 7:   <b>while</b> <math>r \geq m</math> <b>do</b> 8:     <math>r \leftarrow r - m</math> 9:     <math>q \leftarrow q + 1</math> 10:  <math>Q : n = mq + r</math> 11:  <b>return</b> (<math>q, r</math>) </pre>	$\{I \wedge B\} S \{I\}$
--	--------------------------

---

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \textbf{while } B \textbf{ do } S \{Q\}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$$\{I \wedge B\} S \{I\}$$

$$n = mq + r \wedge r \geq m \quad (4)$$

$$r' \leftarrow r - m \quad (5)$$

$$q' \leftarrow q + 1 \quad (6)$$

---


$$n = mq' + r' \quad (7)$$

$$\frac{P' \implies I \quad , \quad \{I \wedge B\} S \{I\} \quad , \quad I \wedge \neg B \implies Q}{\{P'\} \text{ while } B \text{ do } S \{Q\}}$$

$$n = mq + r \wedge r \geq m \quad (1)$$

$$r' \leftarrow r - m \quad (2)$$

$$q' \leftarrow q + 1 \quad (3)$$

---


$$n = mq' + r' \quad (4)$$

$$\begin{aligned} n &= mq' + r' \\ &= m(q + 1) + (r - m) \\ &= mq + r \end{aligned}$$

---

---

```
1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )
```

---

$$I \wedge \neg B \implies Q$$

---

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$$I \wedge \neg B \implies Q$$

$$n = mq + r \quad (1)$$

$$r < m \quad (2)$$

---


$$n = mq + r \quad (3)$$

$$\frac{\{B \wedge t \in D \wedge t = z\} \ S \ \{t \in D \wedge t < z\}}{(\text{while } B \text{ do } S) \text{ will terminate}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$$\frac{\{B \wedge t \in D \wedge t = z\} \ S \ \{t \in D \wedge t < z\}}{(\text{while } B \text{ do } S) \text{ will terminate}}$$

---

```

1: procedure DIVISION( $n, m$ )
2:    $P : n > 0 \wedge m > 0$ 
3:    $r \leftarrow n$ 
4:    $q \leftarrow 0$ 
5:    $P' : r = n \wedge q = 0$ 
6:    $I : n = mq + r$ 
7:   while  $r \geq m$  do
8:      $r \leftarrow r - m$ 
9:      $q \leftarrow q + 1$ 
10:   $Q : n = mq + r$ 
11:  return ( $q, r$ )

```

---

$t = r$

---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---



---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$\text{EUCLID}(m, n) = \gcd(m, n)$$

---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$\text{EUCLID}(m, n) = \gcd(m, n)$$

By mathematical induction on  $n$ , the second parameter.

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$n = 0$$

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$n = 0$$

$$\text{EUCLID}(m, 0) = \text{gcd}(m, 0)$$

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$n = 0$$

$$\text{EUCLID}(m, 0) = \text{gcd}(m, 0)$$

$$m = \text{EUCLID}(m, 0) = \text{gcd}(m, 0) = m$$

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$\text{EUCLID}(m, k) = \text{gcd}(m, k), \forall 0 \leq k \leq n-1 \ (n \geq 1)$$



$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---



---

```

1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )

```

---

$$\text{EUCLID}(m, k) = \text{gcd}(m, k), \forall 0 \leq k \leq n-1 \ (n \geq 1)$$

$$\text{EUCLID}(m, n) = \text{EUCLID}(n, m \bmod n) \quad (1)$$

$$(3)$$

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---



---

```

1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )

```

---

$$\text{EUCLID}(m, k) = \text{gcd}(m, k), \forall 0 \leq k \leq n-1 \ (n \geq 1)$$

$$\text{EUCLID}(m, n) = \text{EUCLID}(n, m \bmod n) \quad (1)$$

$$= \text{gcd}(n, m \bmod n) \quad (2)$$

$$(3)$$

$$\text{EUCLID}(m, n) = \text{gcd}(m, n)$$

---

---

```
1: procedure EUCLID( $m, n$ )
2:   if  $n = 0$  then
3:     return  $m$ 
4:   else
5:     return EUCLID( $n, m \bmod n$ )
```

---

$$\text{EUCLID}(m, k) = \text{gcd}(m, k), \forall 0 \leq k \leq n-1 \ (n \geq 1)$$

$$\text{EUCLID}(m, n) = \text{EUCLID}(n, m \bmod n) \quad (1)$$

$$= \text{gcd}(n, m \bmod n) \quad (2)$$

$$= \text{gcd}(m, n) \quad (3)$$

Jeff Edmonds

# How to Think about Algorithms



CAMBRIDGE

Jeff Edmonds

# How to Think about Algorithms



CAMBRIDGE

$$\{P\} R \{Q\}$$

Jeff Edmonds

# How to Think about Algorithms



$$\{P\} R \{Q\}$$

$$\frac{P \implies I, \{I \wedge B\} S \{I\}, I \wedge \neg B \implies Q}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

Jeff Edmonds

# How to Think about Algorithms



$$\{P\} R \{Q\}$$

$$\frac{P \implies I, \{I \wedge B\} S \{I\}, I \wedge \neg B \implies Q}{\{P\} \text{ while } B \text{ do } S \{Q\}}$$

$$\frac{\{B \wedge t \in D \wedge t = z\} S \{t \in D \wedge t < z\}}{(\text{while } B \text{ do } S) \text{ will terminate}}$$

Thank  
You!





Office 302

Mailbox: H016

hfwei@nju.edu.cn