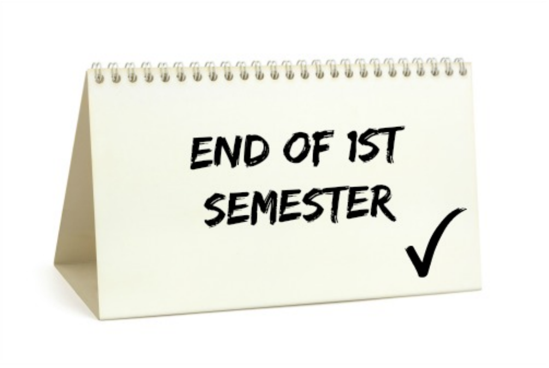
2-1 The Correctness of Algorithms

Hengfeng Wei

hfwei@nju.edu.cn

Feb. 27, 2020



Mathematical Logic   Set Theory   Abstract Algebra

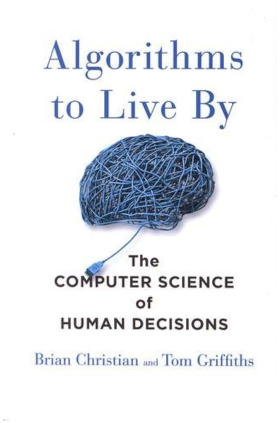
Data Structures Algorithms

Design

Analysis (Time & Space)

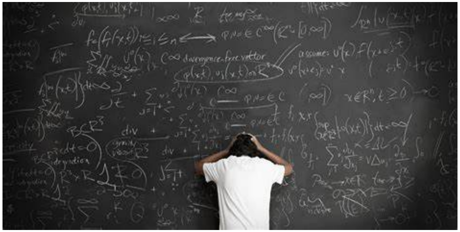
Correctness Proof

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**.



Writing correctness proof needs understanding.

Understanding is hard.

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 $\cdots$

What is the correctness of algorithms?

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

An algorithm is considered correct if it meets its specification.

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

Spec: B is sorted

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

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

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

We use specification languages to specify specs.

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

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.

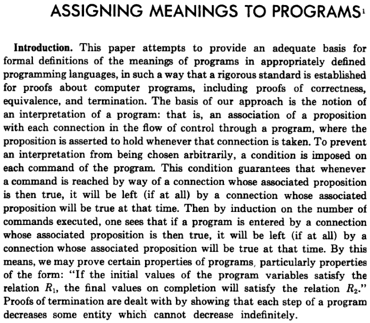
Separating "partial correctness" from "termination"

Definition (Total Correctness)

Total Correctness = Partial Correctness + Termination

- 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.

SHOW ME THE
PROOF
memegenerator.net



ASSIGNING MEANINGS TO PROGRAMS[1]

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.

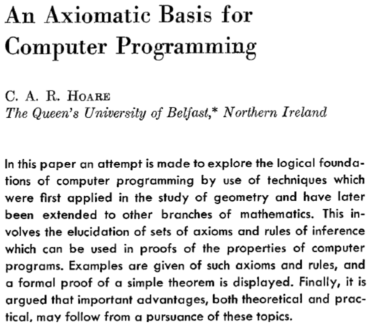
Robert W. Floyd (1936 ~ 2001)

Turing Award (1978)

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

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 logic provides the inference rules.

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

R consists of

- "$x \leftarrow a$"
- "$S; T$"
- "`if` B `then` S `else` T"
- "`while` B `do` S"

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

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

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

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

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

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

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

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

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

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

$$\boxed{I : \mathit{Loop\ Invariant}}$$

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

Q : How to find I?

The general answer is "I don't know."

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

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

Q : How to show its termination?

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

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

$$\boxed{t : \textit{Loop Variant}}$$

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

Q : How to find t?

The general answer is "I don't know."

- $\textsc{reverse}(X)$
- $\textsc{division}(n, m)$
- $\textsc{Euclid}(a, b)$

```
procedure $\textsc{Reverse}(S)$
    $P : \top$
    $X \leftarrow S$                                  $\triangleright$ keep $S$ intact
    $Y \leftarrow \Lambda$                            $\triangleright$ Y is to store the result
    $I : S = \texttt{reverse}(Y) \cdot X$
    while $X \neq \Lambda$ do
        $Y \leftarrow \textsc{Head}(X) \cdot Y$
        $X \leftarrow \textsc{Tail}(X)$
    $Q : Y = \texttt{reverse}(S)$
    return Y
```

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

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

$$P' \implies I$$

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

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

```
1: procedure REVERSE($S$)
2:     $P : \top$
3:     $X \leftarrow S$
4:     $Y \leftarrow \Lambda$
5:     $P' : X = S \land Y = \Lambda$
6:     $I : S = \texttt{reverse}(Y) \cdot X$
7:     while $X \neq \Lambda$ do
8:         $Y \leftarrow \textsc{head}(X) \cdot Y$
9:         $X \leftarrow \textsc{tail}(X)$
10:    $Q : Y = \texttt{reverse}(S)$
11:    return Y
```

$$\{I \land B\}\ S\ \{I\}$$

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

$$Y' \leftarrow \textsc{head}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \textsc{tail}(X) \quad (3)$$

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

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

$$Y' \leftarrow \textsc{head}(X) \cdot Y \quad (2)$$

$$X' \leftarrow \textsc{tail}(X) \quad (3)$$

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

$$\begin{aligned}
S &= \texttt{reverse}(Y') \cdot X' \\
&= \texttt{reverse}(\textsc{head}(X) \cdot Y) \cdot \textsc{tail}(X) \\
&= \texttt{reverse}(Y) \cdot \texttt{reverse}(\textsc{head}(X)) \cdot \textsc{tail}(X) \\
&= \texttt{reverse}(Y) \cdot (\textsc{head}(X) \cdot \textsc{tail}(X)) \\
&= \texttt{reverse}(Y) \cdot X
\end{aligned}$$

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

```
procedure REVERSE($S$)
    $P : \top$
    $X \leftarrow S$
    $Y \leftarrow \Lambda$
    $P' : X = S \land Y = \Lambda$
    $I : S = \texttt{reverse}(Y) \cdot X$
    while $X \neq \Lambda$ do
        $Y \leftarrow \text{HEAD}(X) \cdot Y$
        $X \leftarrow \text{TAIL}(X)$
    $Q : Y = \texttt{reverse}(S)$
    return Y
```

$$I \land \neg B \implies Q$$

$$S = \texttt{reverse}(Y) \cdot X \tag{1}$$

$$X = \Lambda \tag{2}$$

$$Y = \texttt{reverse}(S) \tag{3}$$

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

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

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

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

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

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

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

```
1: procedure DIVISION($(n, m)$)
2:     $P : n > 0 \land m > 0$
3:     $r \leftarrow n$
4:     $q \leftarrow 0$
5:     $P' : r = n \land q = 0$
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 $(q, r)$
```

$$P' \implies I$$

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

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

```
 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 $(q, r)$
```

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

$$n = mq + r \wedge r \geq m \tag{4}$$

$$r' \leftarrow r - m \tag{5}$$

$$q' \leftarrow q + 1 \tag{6}$$

$$n = mq' + r' \tag{7}$$

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

$$n = mq + r \land r \geq m \qquad (1)$$

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

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

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

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

```
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 $(q, r)$
```

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

$$n = mq + r \tag{1}$$

$$r < m \tag{2}$$

$$n = mq + r \tag{3}$$

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

```
 1: procedure DIVISION($(n, m)$)
 2:     $P : n > 0 \land m > 0$
 3:     $r \leftarrow n$
 4:     $q \leftarrow 0$
 5:     $P' : r = n \land q = 0$
 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 $(q, r)$
```

$t = r$

```
procedure EUCLID($m, n$)
    if $n = 0$ then
        return $m$
    else
        return EUCLID($n, m \bmod n$)
```

$$\textsc{Euclid}(m, n) = \texttt{gcd}(m, n)$$

By mathematical induction on n, the second parameter.

$$\textsc{Euclid}(m, n) = \texttt{gcd}(m, n)$$

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

$$n = 0$$

$$\textsc{Euclid}(m, 0) = \texttt{gcd}(m, 0)$$

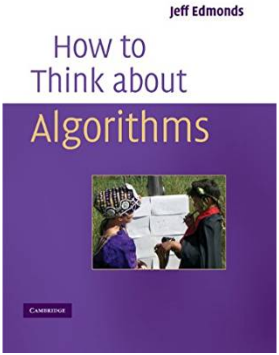
$$m = \textsc{Euclid}(m, 0) = \texttt{gcd}(m, 0) = m$$

$$\textsc{Euclid}(m, n) = \texttt{gcd}(m, n)$$

```
procedure $\textsc{Euclid}(m, n)$
    if $n = 0$ then
        return $m$
    else
        return $\textsc{Euclid}(n, m \bmod n)$
```

$$\textsc{Euclid}(m, k) = \texttt{gcd}(m, k),\ \forall\, 0 \le k \le n - 1\ (n \ge 1)$$

$$\begin{aligned} \textsc{Euclid}(m, n) &= \textsc{Euclid}(n, m \bmod n) && (1) \\ &= \texttt{gcd}(n, m \bmod n) && (2) \\ &= \texttt{gcd}(m, n) && (3) \end{aligned}$$

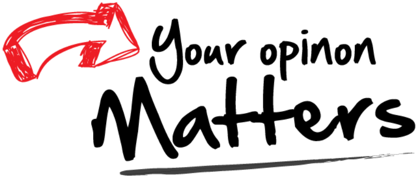


$$\{P\}\ R\ \{Q\}$$

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

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

Thank You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn