

1-4 Algorithms

魏恒峰

hfwei@nju.edu.cn

2019 年 11 月 07 日





DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(a) “for-do” by “while-do”

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(a) “for-do” by “while-do”

```
for (int i = 0; i < N; ++i)
    statement
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(a) “for-do” by “while-do”

```
for (int i = 0; i < N; ++i)
    statement
```

```
int i = 0;
while (i < N)
    statement
    ++i
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(a) “for-do” by “while-do”

```
for (int i = 0; i < N; ++i)
    statement
```

```
int i = 0;
while (i < N)
    statement
    ++i
```

Not General

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(a) “for-do” by “while-do”

```
for (init; cond; inc)
    statement
```

```
init;
while (cond)
    statement
inc
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
```


DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
```

```
while (A)
  B
   $\neg$  A
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
```

```
while (A)
  B
   $\neg$  A // Wrong: side effects?
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
```

```
while (A)
  B
   $\neg$  A // Wrong: side effects?
```

```
flag = 1
while (A && flag)
  B
  flag = 0
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
else
  C
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
else
  C
```

```
flag_if = 1
while (A && flag_if)
  B
  flag_if = 0
flag_else = 1
while ( $\neg$  A && flag_else)
  C
  flag_else = 0
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
else
  C
```

```
flag_if = 1
while (A && flag_if)
  B // Wrong: side effects?
  flag_if = 0
flag_else = 1
while ( $\neg$  A && flag_else)
  C
  flag_else = 0
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
else
  C
```

```
flag_if = 1
while (A && flag_if)
  B // Wrong: side effects?
  flag_if = 0
flag_else = 1
while ( $\neg$  A && flag_else)
  C
  flag_else = 0
```

```
flag = 1
while (A && flag)
  B
  flag = 0

while ( $\neg$  A && flag)
  C
  flag = 0
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(b) “if-then & if-then-else” by “while-do”

```
if (A)
  B
else
  C
```

```
flag_if = 1
while (A && flag_if)
  B // Wrong: side effects?
  flag_if = 0
flag_else = 1
while ( $\neg$  A && flag_else)
  C
  flag_else = 0
```

```
flag = 1
while (A && flag)
  B
  flag = 0
//  $\neg$ A not necessary
while ( $\neg$  A && flag)
  C
  flag = 0
```


DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(c) “while-do” by “if-then & goto”

(d) “while-do” by “repeat-until & if-then”

```
while (A)
  B
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(c) “while-do” by “if-then & goto”

(d) “while-do” by “repeat-until & if-then”

```
while (A)
    B
```

```
L: if (A)
    B
    goto L
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(c) “while-do” by “if-then & goto”

(d) “while-do” by “repeat-until & if-then”

```
while (A)
  B
```

```
L: if (A)
    B
    goto L
```

```
if (A)
  repeat
    B
  until ( $\neg$  A)
```

DH 2.5: Simulations

Perform the following simulations of some control constructs by others.

(c) “while-do” by “if-then & goto”

(d) “while-do” by “repeat-until & if-then”

```
while (A)
  B
```

```
L: if (A)
    B
    goto L
```

```
if (A) // 'if' in 'repeat'?
  repeat
    B
  until ( $\neg$  A)
```

DH 2.8: Simulations

Simulate “while-do” by “if-then-else & recursive”.

```
while (A)  
  B
```

DH 2.8: Simulations

Simulate “while-do” by “if-then-else & recursive”.

```
while (A)
  B
```

```
simulateWhile() {
  if (A)
    B
  simulateWhile();
}
```

DH 2.8: Simulations

Simulate “while-do” by “if-then-else & recursive”.

```
while (A)  
    B
```

```
simulateWhile() { // define function  
    if (A)  
        B  
        simulateWhile();  
}
```

- (1) A;B
- (2) if-then
- (3) if-then-else
- (4) for-do
- (5) while-do
- (6) repeat-until

- (1) A;B
- (2) if-then
- (3) if-then-else
- (4) for-do
- (5) while-do
- (6) repeat-until

```
repeat  
  B  
until ( $\neg$  A)
```

- (1) A;B
- (2) if-then
- (3) if-then-else
- (4) for-do
- (5) while-do
- (6) repeat-until

```
repeat  
  B  
until ( $\neg$  A)
```

```
B  
while (A)  
  B
```

- (1) A;B
- (2) if-then
- (3) if-then-else
- (4) for-do
- (5) while-do
- (6) repeat-until

```
repeat  
  B  
until ( $\neg$  A)
```

```
B  
while (A)  
  B
```

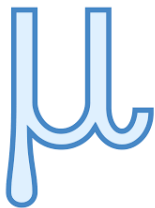
Theorem (“On Folk Theorems” (David Harel, 1980))

*Any **computable function** can be computed by a “while-do” (and “;”) program (with additional Boolean variables).*





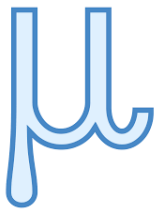
What are “Computable Functions”?



μ -Recursive Functions (1931-1934)



Kurt Gödel (1906-1978)



μ -Recursive Functions
(1931-1934)



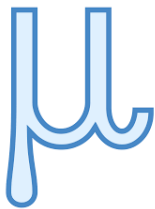
Kurt Gödel (1906-1978)



Lambda Calculus
(1933-1935)



Alonzo Church (1903-1995)



μ -Recursive Functions
(1931-1934)



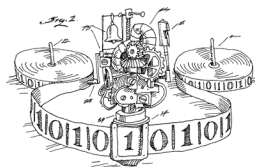
Kurt Gödel (1906-1978)



Lambda Calculus
(1933-1935)



Alonzo Church (1903-1995)

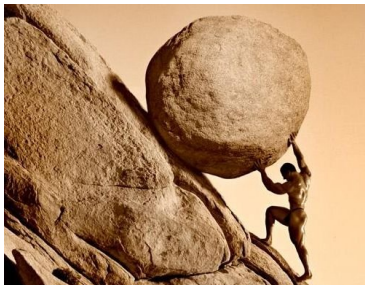


Turing Machines
(1936-1937)

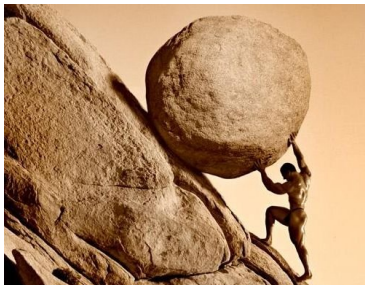


Alan Turing (1912-1954)

Bounded Iterations *vs.* Unbounded Iterations



Bounded Iterations *vs.* Unbounded Iterations



Q : Why unbounded iterations?

Definition (Primitive Recursive)

$$\begin{cases} h(0) &= k, \\ h(t+1) &= g(t, h(t)) \end{cases}$$

Definition (Primitive Recursive)

$$\begin{cases} h(0) &= k, \\ h(t+1) &= g(t, h(t)) \end{cases}$$

$$\begin{cases} h(x_1, \dots, x_n, 0) &= f(x_1, \dots, x_n), \\ h(x_1, \dots, x_n, t+1) &= g(t, h(x_1, \dots, x_n, t), x_1, \dots, x_n) \end{cases}$$

Definition (Primitive Recursive)

$$\begin{cases} h(0) &= k, \\ h(t+1) &= g(t, h(t)) \end{cases}$$

$$\begin{cases} h(x_1, \dots, x_n, 0) &= f(x_1, \dots, x_n), \\ h(x_1, \dots, x_n, t+1) &= g(t, h(x_1, \dots, x_n, t), x_1, \dots, x_n) \end{cases}$$

$$\forall/\exists t \leq y : P(t, x_1, \dots, x_n)$$



Theorem (Ackermann Function)

The Ackermann function is μ -recursive but not primitive recursive.



Theorem (Ackermann Function)

The *Ackermann function* is *μ -recursive* but not *primitive recursive*.

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$



Theorem (Ackermann Function)

The *Ackermann function* is *μ -recursive* but not *primitive recursive*.

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

$$A(4, 2) = ?$$



Theorem (Ackermann Function)

The *Ackermann function* is *μ -recursive* but not *primitive recursive*.

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

$$A(4, 2) = ?$$

$$f(x) \triangleq \mu_y (g(x, y)) \triangleq \left(\underset{y}{\operatorname{argmin}} g(x, y) = 0 \right)$$

Thank
You!