1-4 Algorithms

魏恒峰

hfwei@nju.edu.cn

2019 年 11 月 07 日


IS THIS A
SIMULATION?

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
```

```
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) // not general!
  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 (init; cond; inc)
  statement
```

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

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
```

Whether to use "while" or "for" is largely a matter of personal preference.

— *K&R C Bible*

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
¬ 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
¬ 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
  ¬ 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 (¬ 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 (¬ 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 (¬ A && flag_else)
  C
  flag_else = 0
```

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

while (¬ 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 (¬ A && flag_else)
  C
  flag_else = 0
```

```
flag = 1
while (A && flag)
  B
  flag = 0
// ¬A not necessary
while (¬ 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 (¬ 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)  // no ''if''?
  repeat
    B
  until (¬ 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();

  return;
}
```

DH 2.8: Simulations

Simulate "`while-do`" by "`if-then-else` & `recursive`".

```
while (A)
  B
```

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

  return;
}
```

DH 2.8: Simulations

Simulate "`while-do`" by "`if-then-else` & `recursive`".

```
while (A)
  B
```

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

  return;
}
```

(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 (¬ A)
```

(1) A;B

(2) if-then

(3) if-then-else

(4) for-do

(5) while-do

(6) repeat-until

```
repeat
  B
until (¬ 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 (¬ A)
```

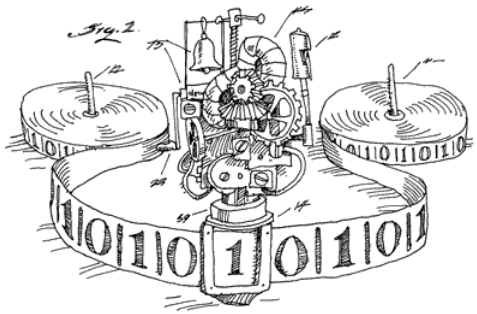
```
B
while (A)
  B
```

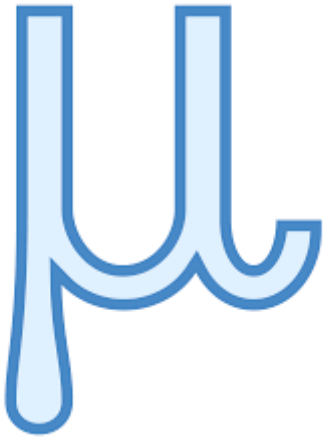
Theorem ("On Folk Theorems" (David Harel, 1980))

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

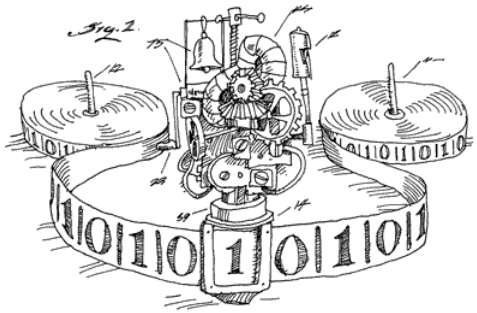

i minimalism


i minimalism





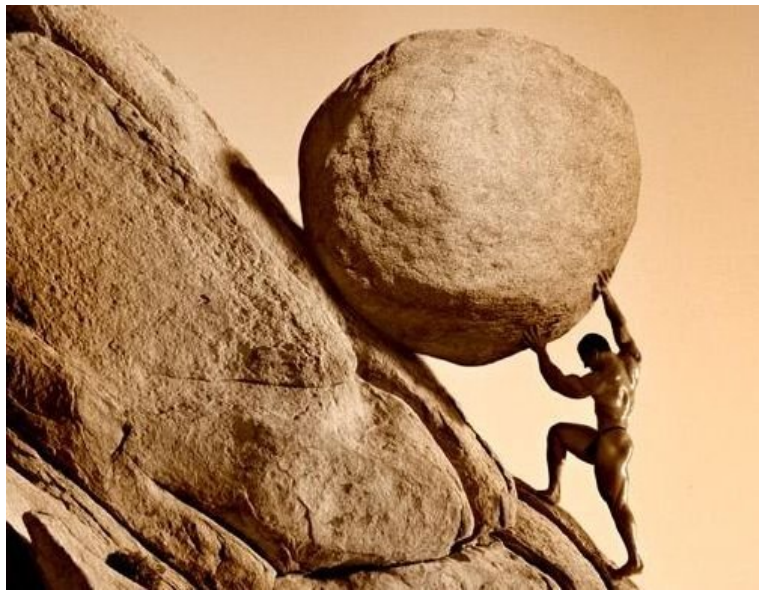
Simulations for Equivalence



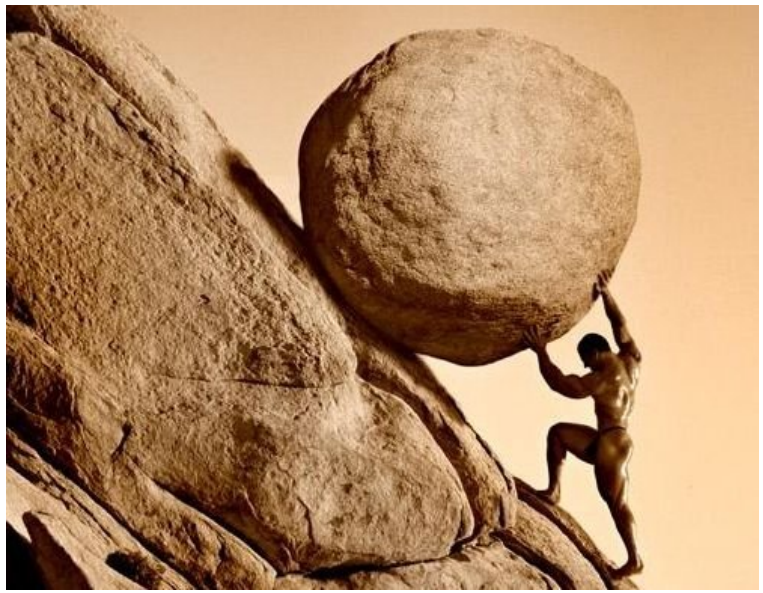
λ

μ

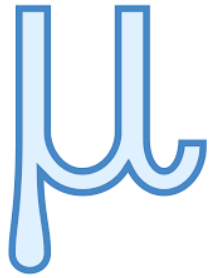
Bounded Iterations *vs.* Unbounded Iterations



Bounded Iterations *vs.* Unbounded Iterations

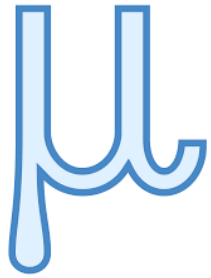


Q : Why unbounded iterations?



μ-Recursive Functions

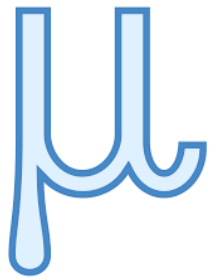
$$\mu y\big(g(x,y)\big) = \Big(\operatorname*{argmin}_{y} g(x,y) = 0\Big)$$



μ-Recursive Functions

$$\mu y\big(g(x,y)\big) = \Big(\operatorname*{argmin}_{y} g(x,y) = 0\Big)$$

Unbounded iterations: "`while-do`"



μ-Recursive Functions

$$\mu y(g(x,y)) = \left(\operatorname*{argmin}_y g(x,y) = 0\right)$$

Unbounded iterations: "`while-do`"

Theorem (Ackermann Function)

The Ackermann function is μ-recursive but not primitive recursive (which contains bounded iterations.).


Thank
You!