CS 511, Fall 2018, Handout 12

Unwinding Programs: Two Small Examples

Assaf Kfoury

20 September 2018

(Last modified: 24 September 2018)

GCD function in Python

original GCD program

```
1 :  def gcd(x,y) :
2 :      while x != y
3 :          if x > y :
4 :              x = x - y
5 :          else :
6 :              y = y - x;
7 :      return x
```

GCD function in Python

original GCD program

```
1 : def gcd(x,y) :
2 :     while x != y
3 :         if x > y :
4 :             x = x - y
5 :         else :
6 :             y = y - x;
7 :     return x
```

unwinding GCD program

```
1 : def gcd(x,y) :

2 :     if x == y : return x
3 :     else :    if x > y :
4 :                   x = x - y
5 :               else :
6 :                   y = y - x;

2 :     if x == y : return x
3 :     else :    if x > y :
4 :                   x = x - y
5 :               else :
6 :                   y = y - x;

2 :     if x == y : return x
3 :     else :    if x > y :
4 :                   x = x - y
5 :               else :
6 :                   y = y - x;

 .
 .
 .      (forever!)
```

GCD function in Python

original GCD program

```
1 : def gcd(x,y) :
2 :     while x != y
3 :         if x > y :
4 :             x = x - y
5 :         else :
6 :             y = y - x;
7 :     return x
```

unwinding GCD program

```
1 : def gcd(x,y) :

2 :     if x == y : return x
3 :     else :    if x > y :
4 :                   x = x - y
5 :               else :
6 :                   y = y - x;

2 :     if x == y : return x
3 :     else :    if x > y :
4 :                   x = x - y
5 :               else :
6 :                   y = y - x;

2 :     if x == y : return x
3 :     else :    if x > y :
4 :                   x = x - y
5 :               else :
6 :                   y = y - x;

 .
 .
 .
          (forever!)
```

the finite execution paths can be described by the regular expression : $1\ 2\ \left(\ 3\ (\ 4\ +\ 5\ 6\ )\ 2\ \right)^*\ 7$

Analysis of the GCD function

For each finite execution path π in the program, we can write a wff φ_π that specifies:

(a) conditions (on the inputs) under which path π is followed by the program,

(b) the output in case the conditions in (a) are satisfied.

Analysis of the GCD function

For each finite execution path π in the program, we can write a wff φ_π that specifies:

(a) conditions (on the inputs) under which path π is followed by the program,

(b) the output in case the conditions in (a) are satisfied.

- for path $\pi \triangleq 1\ 2\ 7$:

  $\varphi_\pi \triangleq (x = y) \rightarrow$ **return** x
- for path $\pi \triangleq 1\ 2\ 3\ 4\ 2\ 7$:

  $\varphi_\pi \triangleq (x > y) \wedge (x = 2y) \rightarrow$ **return** y
- for path $\pi \triangleq 1\ 2\ 3\ 5\ 6\ 2\ 7$:

  $\varphi_\pi \triangleq (x < y) \wedge (y = 2x) \rightarrow$ **return** x
- for path $\pi \triangleq 1\ 2\ 3\ 4\ 2\ 3\ 4\ 2\ 7$:

  $\varphi_\pi \triangleq (x > y) \wedge (x > 2y) \wedge (x = 3y) \rightarrow$ **return** y
- for path $\pi \triangleq 1\ 2\ 3\ 4\ 2\ 3\ 5\ 6\ 2\ 7$:

  $\varphi_\pi \triangleq (x > y) \wedge (x < 2y) \wedge (2x = 3y) \rightarrow$ **return** $(x - y)$
- etc.

Analysis of the GCD function

For each finite execution path π in the program, we can write a wff φ_π that specifies:

(a) conditions (on the inputs) under which path π is followed by the program,

(b) the output in case the conditions in (a) are satisfied.

- for path $\pi \triangleq 1\ 2\ 7$:

  $\varphi_\pi \triangleq (x = y) \rightarrow$ **return** x
- for path $\pi \triangleq 1\ 2\ 3\ 4\ 2\ 7$:

  $\varphi_\pi \triangleq (x > y) \wedge (x = 2y) \rightarrow$ **return** y
- for path $\pi \triangleq 1\ 2\ 3\ 5\ 6\ 2\ 7$:

  $\varphi_\pi \triangleq (x < y) \wedge (y = 2x) \rightarrow$ **return** x
- for path $\pi \triangleq 1\ 2\ 3\ 4\ 2\ 3\ 4\ 2\ 7$:

  $\varphi_\pi \triangleq (x > y) \wedge (x > 2y) \wedge (x = 3y) \rightarrow$ **return** y
- for path $\pi \triangleq 1\ 2\ 3\ 4\ 2\ 3\ 5\ 6\ 2\ 7$:

  $\varphi_\pi \triangleq (x > y) \wedge (x < 2y) \wedge (2x = 3y) \rightarrow$ **return** $(x - y)$
- etc.

Remarks :

- φ_π is not a PL wff
- we have not defined a systematic way of writing φ_π (tricky!)
- so far we have not used timestamps

Analysis of the GCD function

unwinding GCD program with timestamps

```
1 :  def gcd(x0, y0) :

2 :      if x0 == y0 : return x0
3 :      else :   if x0 > y0 :
4 :                   x1 = x0 − y0 ; y1 = y0
5 :               else :
6 :                   y1 = y0 − x0 ; x1 = x0

2 :      if x1 == y1 : return x1
3 :      else :   if x1 > y1 :
4 :                   x2 = x1 − y1 ; y2 = y1
5 :               else :
6 :                   y2 = y1 − x1 ; x2 = x1

2 :      if x2 == y2 : return x2
3 :      else :   if x2 > y2 :
4 :                   x3 = x2 − y2 ; y3 = y2
5 :               else :
6 :                   y3 = y2 − x2 ; x3 = x2

 ⋮
         (forever!)
```

(instructions in red are new)

Analysis of the GCD function

unwinding GCD program with timestamps

```
1 :  def gcd(x0, y0) :

2 :      if x0 == y0 : return x0
3 :      else :   if x0 > y0 :
4 :                   x1 = x0 − y0 ; y1 = y0
5 :               else :
6 :                   y1 = y0 − x0 ; x1 = x0

2 :      if x1 == y1 : return x1
3 :      else :   if x1 > y1 :
4 :                   x2 = x1 − y1 ; y2 = y1
5 :               else :
6 :                   y2 = y1 − x1 ; x2 = x1

2 :      if x2 == y2 : return x2
3 :      else :   if x2 > y2 :
4 :                   x3 = x2 − y2 ; y3 = y2
5 :               else :
6 :                   y3 = y2 − x2 ; x3 = x2

 ⋮
         (forever!)
```

(instructions in red are new)

Exercise: For each finite execution path π in the unwound program with timestamps, write a wff φ_π specifying:

(a) conditions (on the inputs) under which path π is followed by the program,

(b) the output in case the conditions in (a) are satisfied.

Another small example

original program `foo`

```
1 : def foo(x,y) :
2 :     if x < y :
3 :         x = x+y
4 :     else :
5 :         x = x;
6 :     for i from 1 to 3
7 :         y = x+y+1;
8 :     return x+y;
```

Another small example

original program `foo`

```
1 :  def foo(x, y) :
2 :      if x < y :
3 :          x = x + y
4 :      else :
5 :          x = x;
6 :      for i from 1 to 3
7 :          y = x + y + 1;
8 :      return x + y;
```

unwinding program `foo` with timestamps

```
1 :    def foo(x0, y0) :
2 :        if x0 < y0 :
3 :            x1 = x0 + y0
4 :        else :
5 :            x1 = x0;
6-7 :      y1 = x1 + y0 + 1;
6-7 :      y2 = x1 + y1 + 1;
6-7 :      y3 = x1 + y2 + 1;
8 :        return x1 + y3;
```

Another small example

original program `foo`

```
1 :  def foo(x,y) :
2 :      if x < y :
3 :          x = x+y
4 :      else :
5 :          x = x;
6 :      for i from 1 to 3
7 :          y = x+y+1;
8 :      return x+y;
```

unwinding program `foo` with timestamps

```
1 :    def foo(x0,y0) :
2 :        if x0 < y0 :
3 :            x1 = x0+y0
4 :        else :
5 :            x1 = x0;
6-7 :      y1 = x1+y0+1;
6-7 :      y2 = x1+y1+1;
6-7 :      y3 = x1+y2+1;
8 :        return x1+y3;
```

Only two execution paths π and two wff's φ_π, where "=" is now *equality*, not *update*:

- for path $\pi_1 \triangleq 1\ 2\ 3\ (6\ 7)^3\ 8$:

$$\varphi_{\pi_1} \triangleq (x_0 < y_0) \rightarrow (x_1 = x_0 + y_0) \wedge (y_1 = x_1 + y_0 + 1) \wedge (y_2 = x_1 + y_1 + 1) \wedge (y_3 = x_1 + y_2 + 1) \wedge \mathbf{out} = x_1 + y_3$$

- for path $\pi_2 \triangleq 1\ 2\ 4\ 5\ (6\ 7)^3\ 8$:

$$\varphi_{\pi_2} \triangleq \neg(x_0 < y_0) \rightarrow (x_1 = x_0) \wedge (y_1 = x_1 + y_0 + 1) \wedge (y_2 = x_1 + y_1 + 1) \wedge (y_3 = x_1 + y_2 + 1) \wedge \mathbf{out} = x_1 + y_3$$

(THIS PAGE INTENTIONALLY LEFT BLANK)