EECS, Lassonde School of Engineering, York

Mysterious Code & Specifications

Jonathan S. Ostroff
Software Engineering Laboratory

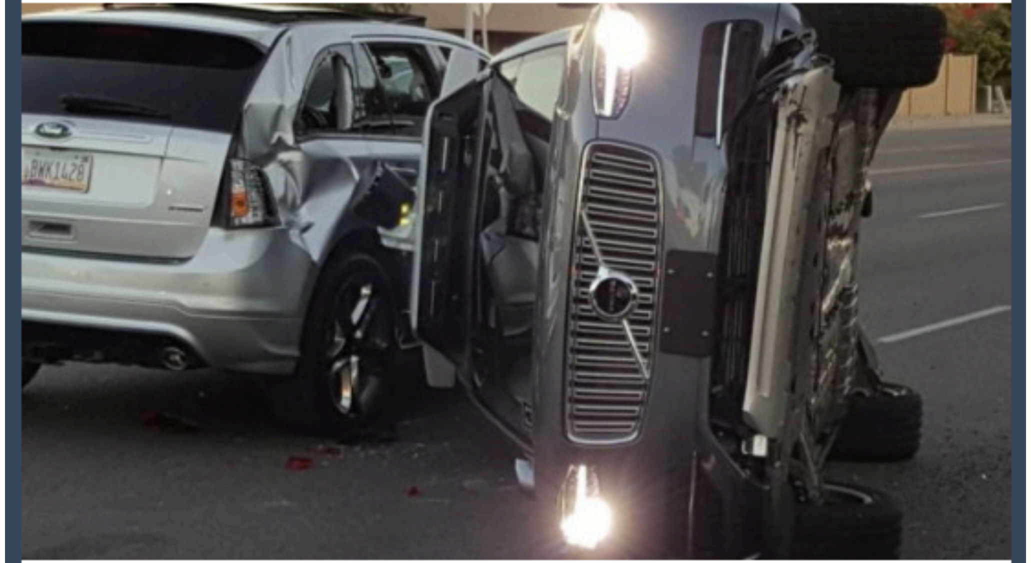
Uber suspends self-driving cars after Arizona crash

Uber suspends self-driving cars after Arizona crash

BBC 28 March 2017

BBC

http://www.bbc.com/news/technology-39397211



A self-driven Volvo SUV owned and operated by Uber Technologies Inc. is flipped on its side after a collision in Tempe, Arizona, U.S. on March 24, 2017.

Uber said the car was in self-driving mode at the time of the crash
Uber has pulled its self-driving cars from the roads after an accident which left one of the vehicles on its side.

```python
# mystery routine
def even(number):
        return number % 2 == 0
def odd(number):
        return number % 2 != 0
a = 4
b = 5
x = a
y = b
z = 0
while (y > 0 and even(y)) or odd(y):
        if y > 0 and even(y):
                x, y = x + x, y // 2
        else:
                y, z = y - 1, z + x
print(z)
```

```
q(a, b: INTEGER): INTEGER
        -- mystery routine?
    local
        x, y, z: INTEGER
    do
        from
            x := a; y := b; z := 0
        until
            not (y > 0 and even(y)) and not odd(y)
        loop
            if y > 0 and even(y) then
                y := y // 2; x := x + x
            else -- odd(y)
                y := y - 1; z := z + x
            end
        end
        Result := z
    end
```

Here is more information. What does this routine do?

```
q(a, b: INTEGER): INTEGER
        --comment??
    require
        b >= 0
    local
        x, y, z: INTEGER
    do
        from
            x := a; y := b; z := 0
        invariant
            ????

        until
            not (y > 0 and even(y)) and not odd(y)
        loop
            if y > 0 and even(y) then
                y := y // 2; x := x + x
            else
                check odd(y) end
                y := y - 1; z := z + x
            end
        variant
            ????
        end
        Result := z
    ensure
        ????
    end
```

```
q(a, b: INTEGER): INTEGER
        --comment??
    require
        b >= 0
    local
        x, y, z: INTEGER
    do
        from
            x := a; y := b; z := 0
        invariant
            ????

        until
            not (y > 0 and even(y)) and not odd(y)
        loop
            if y > 0 and even(y) then
                y := y // 2; x := x + x
            else
                check odd(y) end
                y := y - 1; z := z + x
            end
        variant
            ????
        end
        Result := z
    ensure
        ????
    end
```

```
q(a, b: INTEGER): INTEGER
        --comment??
    require
        b >= 0
    local
        x, y, z: INTEGER
    do
        from
            x := a; y := b; z := 0
        invariant
            ????

        until
            not (y > 0 and even(y)) and not odd(y)
        loop
            if y > 0 and even(y) then
                y := y // 2; x := x + x
            else
                check odd(y) end
                y := y - 1; z := z + x
            end
        variant
            ????
        end
        Result := z
    ensure
        ????
    end
```

```
q(a, b: INTEGER): INTEGER
        -- Result is `a' multiplied by `b'
    require
        b >= 0
    local
        x, y, z: INTEGER
    do
        from
            x := a; y :=
        invariant
            y >= 0
            z + x*y = a*b
        until
            not (y > 0 an
        loop
            if y > 0 and even(y) then
                y := y // 2; x := x + x
            else
                check odd(y) end
                y := y - 1; z := z + x
            end
        variant
            y
        end
        Result := z
    ensure
        Result = a*b
    end
```

```
q (a, b: INTEGER_32): INTEGER_32
        -- Result is `a' multiplied by `b'
    require
            b >= 0
    ensure
            Result = a * b
```

```
q (a, b: INTEGER_32): INTEGER_32
        -- Result is `a' multiplied by `b'
    require
            b >= 0
    ensure
            Result = a * b
```

- Some embedded CPUs do not have a multiplier
- Also needed for arbitrary precision multiplication on 32 bit processors
- a*b = a + a + a … linear in b
- Whereas the q algorithm does it in log(b) time

Weakest Precondition Calculus

```
wp(“x:=e”,R) ≙ WD(e)∧R[x:e] -- also simultaneous assign
                     -- WD(e) means expression e is well defined

wp(“S1;S2”) ≙ wp(“S1”,wp(“S2”,R)

wp(“if B then S1 else S2”,R)
≙ WD(B) ∧(B ⇒ wp(“S1”,R)) ∧(¬B ⇒ wp(“S2”,R))

Thus: x:=a; y:=b; z:=0 can be reduced to the simultaneous
assignment:  x, y, z := a, b, 0

{Q} program {R}  ≙  Q ⇒ wp(“program”,R)
```

Weakest Precondition Calculus

```
r(a:T) -- routine r
  require P
  do
    from init
    invariant I
    until B
    do
      body
    variant V
    end
  ensure R
  end
```

Proof Obligations for loop

1. $\{P\}$ init $\{I\}$
2. $\{I \wedge \neg B\}$ body $\{I\}$
3. $(I \wedge B) \Rightarrow R$
4. $(I \wedge \neg B) \Rightarrow (V >= 0)$
5. $\{I \wedge \neg B \wedge V=V_0\}$ body $\{V<V_0\}$

Note: This also means that:

V < 0 implies (not I) or B

(i.e. we have terminated)

(1) $Q \Rightarrow wp(\text{“x, y, z := a, b, 0”}, I)$

$I: y \geq 0 \wedge z + x*y = a*b$

```
q(a, b: INTEGER): INTEGER
        -- Result is `a' multiplied by `b'
    require
        b >= 0
    local
        x, y, z: INTEGER
    do
        from
            x := a; y := b; z := 0
        invariant
            y >= 0
            z + x*y = a*b
        until
            not (y > 0 and even(y)) and not odd(y)
        loop
            if y > 0 and even(y) then
                y := y // 2; x := x + x
            else
                check odd(y) end
                y := y - 1; z := z + x
            end
        variant
            y
        end
        Result := z
    ensure
        Result = a*b
    end
```

wp("x, y, z := a, b, 0", I)

= « definition of wp for assignment »

$I\ [x, y, z := a, b, 0]$

= « substitution »

$b \geq 0 \wedge 0 + a*b = a*b$

= « arithmetic »

$b \geq 0$

= « definition of Q»

Q

{I ∧ ¬B ∧ B1} S1 {I}
= {I ∧ B1} S1 {I}

I: y ≥ 0 ∧ z + x*y = a*b

```
q(a, b: INTEGER): INTEGER
        -- Result is `a' multiplied by `b'
    require
        b >= 0
    local
        x, y, z: INTEGER
    do
        from
            x := a; y := b; z := 0
        invariant
            y >= 0
            z + x*y = a*b
        until
            not (y > 0 and even(y)) and not odd(y)
        loop
            if y > 0 and even(y) then
                y := y // 2; x := x + x
            else
                check odd(y) end
                y := y - 1; z := z + x
            end
        variant
            y
        end
        Result := z
    ensure
        Result = a*b
    end
```

wp("x := x + x ‖ y := y÷2", I)
= « definition of wp for assignment »
I[x := x + x ‖ y := y ÷ 2]
= « substitution »
y÷2 ≥ 0 ∧ z + (x+x)*(y÷2) = a*b
= « assume B1: y > 0 ∧ even(y),
arithmetic (x+x)*(y÷2) = 2x*(y÷2) = x*y »
y ≥ 0 ∧ z + x*y = a*b
= « defn. of I »
I

Thus I ∧ B1 ⇒ wp("x := x + x ‖ y := y ÷ 2", I),
i.e. {I ∧ B1} S1 {I}

{I ∧ ¬B ∧ B2} S2 {I}
= {I ∧ B2} S2 {I}

I: y ≥ 0 ∧ z + x*y = a*b

wp(“y := y-1‖ z := z+x", I)
= « definition of wp for assignment »
I[y := y-1 ‖ z := z+x]
= « substitution »
(y-1≥0) ∧ (z+x + x*(y-1) = a*b)
= « assume B2: odd(y), i.e. y≠0,
arithmetic: z+x + x*(y-1) = z + x*y»
(odd(y)∧y≥1) ∧ (z + x*y = a*b)
= « arithmetic: odd(y)∧y≥1 ≡ odd(y)∧y>=0 »
I ∧ B2

```
q(a, b: INTEGER): INTEGER
    -- Result is `a' multiplied by `b'
  require
    b >= 0
  local
    x, y, z: INTEGER
  do
    from
      x := a; y := b; z := 0
    invariant
      y >= 0
      z + x*y = a*b
    until
      not (y > 0 and even(y)) and not odd(y)
    loop
      if y > 0 and even(y) then
        y := y // 2; x := x + x
      else
        check odd(y) end
        y := y - 1; z := z + x
      end
    variant
      y
    end
    Result := z
  ensure
    Result = a*b
  end
```

Thus, {I ∧ B2} S2 {I}, i.e. I ∧ B2 ⇒ wp(“S2”, R)

```
5. {P ∧ Bi ∧ t=T} Si {t < T}
=================================
 P is: y ≥ 0 ∧ z + x*y = a*b
 B1 is: y > 0 ∧ even(y)

   wp("x := x + 2 ‖ y := y ÷ 2", y < T)
 = « substitution »
   y ÷ 2 < T

  P ∧ Bi ∧ t=T
 = « definitions »
  y ≥ 0 ∧ z + x*y = a*b ∧ y > 0 ∧ even(y) ∧ y = T
⇒ « arithmetic »
  y > 0 ∧ even(y) ∧ y = T
⇒  « arithmetic: (y÷2) < y »
  y÷2 < T
⇒ < defn. of wp above>
  wp("x := x + 2 ‖ y := y ÷ 2", y < T)

 Thus, {P ∧ Bi ∧ t=T} Si {t < T}
```

Dijkstra's non-deterministic guarded command language

```
do  B1 → S1
[]  B2 → S2
od
```

$BB = B1 \vee B2$

So long as BB holds,
choose a guarded command
$Bi \rightarrow Si$
and execute it

Dijkstra's guarded command language

$BB = B1 \lor B2$

```
{P}
do  {P ∧ B1} B1 → S1 {P}
[]  {P ∧ B2} B2 → S2 {P}
od
{P ∧ ¬BB}
```

The algorithm in Dijkstra's guarded command language

```
BB = B1 ∨ B2

x, y, z := a, b, 0

do y > 0 ∧ even(y) → x := x + x ‖ y := y ÷ 2
[] odd(y)          → y := y - 1 ‖ z := z + x
od
```

The algorithm in Dijkstra’s guarded command language

```
BB = B1 ∨ B2

{Q: b ≥ 0}
x, y, z := a, b, 0
{P: y ≥ 0 ∧ z + x*y = a*b}
{variant t: y}
do y > 0 ∧ even(y) → {P ∧ B1} x := x + x ‖ y := y ÷ 2 {P}
[] odd(y)          → {P ∧ B2} y := y - 1 ‖ z := z + x {P}
od
{P ∧ ¬BB}
{R: z = a*b}
```

Dijkstra's guarded command language

```
BB = B1 ∨ B2

{Q}
init
{invariant: P}
{variant: t}
do {P ∧ B1} B1 → S1 {P}
[] {P ∧ B2} B2 → S2 {P}
od
{P ∧ ¬BB}
{R}
```

Prove {Q} loop {R}

1. Prove that P is true initially, $Q \Rightarrow P$
2. Prove that each guarded command preserves P, i.e. {P ∧ Bi} Si {P}
3. Prove that the invariant entails the postcondition, i.e. $P \wedge \neg B \Rightarrow R$
4. Show that variant is bounded below $P \wedge BB \Rightarrow t >= 0$
5. Show that the variant decreases each iteration {P ∧ Bi ∧ t=T} Si {t < T}