

Task 0

```
$ wget hexgolems.com/smt/img.ova
```

setup VM

```
$ ssh smtworkshop@127.0.0.1 -p 2222
```

```
password: hnpsmtworkshop
```



SMT-Solver

for reversing



About me

coco@hexgolems.de

PHD student from Ruhr-Universität Bochum
RE, security, theory and bouldering





SMT Solver



$x < 5$
 $x + 3 \geq 4$
 $x * 2 < 3$
 $x > 0$



Constraints



SMT Solver



$$\begin{aligned}x &< 5 \\ x+3 &\geq 4 \\ x*2 &< 3 \\ x &> 0\end{aligned}$$

Constraints



SMT Solver



$$x = 1$$

Solution



$$\begin{aligned}x &< 5 \\x + 3 &\geq 4 \\x * 2 &< 3 \\x &> 0\end{aligned}$$

Constraints



Magic

$$x = 1$$

Solution



Why bother?



Why bother?

ENC("secret...", k) == "13b7c9...."



Why bother?

ENC("secret...", k) == "13b7c9...."
(Happened to Petya)



Why bother?

ENC("secret...", k) == "13b7c9...."

Or any function



CONSTRAINTS



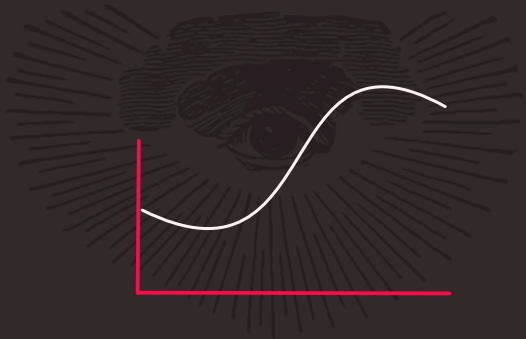
Linear Constraints

$$x + y == z$$



Non-Linear Constraints

$$x * y == z$$



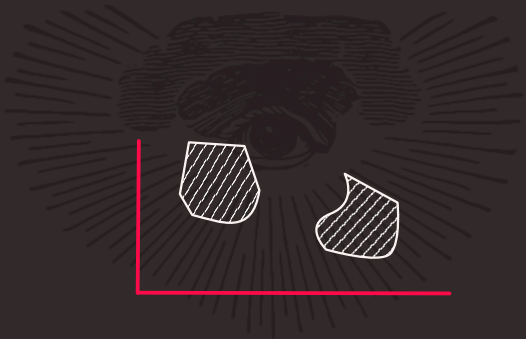
Inequalities

$$x * y \leq z$$



Logical Ops

$(x * y \leq z) \parallel x < 4$



Binary Ops

$x \& y \leq z$

???



Whatever goes
here

```
if( {  
    .....  
} ) {  
    ...  
}
```



We can use as
constraint^{*}

Assert()



*Standard Terms & Conditions Apply

(u)int ✓

$$x * 2 == 4$$

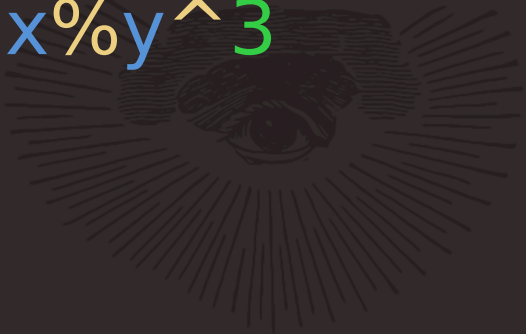


Overflows ✓

$x-1 > 5 \ \&\& \ x < 5$



Weird Ops ✓

$$x^0 y^3$$


float

$$x * x == 2$$



~~float~~

$$\cancel{x * x = 2}$$



int[] ✓

x[y] + y



CODE



```
Assert( x*20 == 100 )  
Assert( x <= 100 )
```



```
Assert( x*20 == 100 )  
Assert( x <= 100 )
```

What is x ?



```
x = Var(64, "x")  
Assert( x*20 == 100 )  
Assert( x <= 100 )
```



Unique

```
x = Var(64, "x")  
Assert( x*20 == 100 )  
Assert( x <= 100 )
```



```
x = Var(64, "x")  
Assert( x*20 == 100 )  
Assert( x <= 100 )
```

Signedness?



```
x = Var(64, "x")  
Assert( x*20 == 100 )  
Assert(  Ute(x, 100) )
```




```
b = Boolector()  
x = b.Var(64, "x")  
b.Assert( x*20 == 100 )  
b.Assert( b.Ulte(x, 100) )
```



```
from boolector import Boolector
```

```
b = Boolector()
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

```
if res == b.SAT:
    print("{} {}".format(x.symbol, int(x.assignment, 2)))
else:
    print("unsat")
```



Import

```
from boolector import Boolector
```

```
b = Boolector()  
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)  
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)  
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

```
if res == b.SAT:  
    print("{} {}".format(x.symbol, int(x.assignment, 2)))  
else:  
    print("unsat")
```



```
from boolector import Boolector
```

```
b = Boolector()  
b.Set_opt("model_gen", 1)
```

We want the solution

```
const = b.Const(133713378, 64)  
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)  
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

```
if res == b.SAT:  
    print("{} {}".format(x.symbol, int(x.assignment, 2)))  
else:  
    print("unsat")
```



```
from boolector import Boolector
```

```
b = Boolector()
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

```
if res == b.SAT:
    print("{} {}".format(x.symbol, int(x.assignment, 2)))
else:
    print("unsat")
```

We can use constants



```
from boolector import Boolector
```

```
b = Boolector()
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

```
if res == b.SAT:
    print("{} {}".format(x.symbol, int(x.assignment, 2)))
else:
    print("unsat")
```

With given value



```
from boolector import Boolector
```

```
b = Boolector()  
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)  
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)  
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

```
if res == b.SAT:  
    print("{} {}".format(x.symbol, int(x.assignment, 2)))  
else:  
    print("unsat")
```

... and bit width



```
from boolector import Boolector
```

```
b = Boolector()  
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)  
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)  
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

Try to solve

```
if res == b.SAT:  
    print("{} {}".format(x.symbol, int(x.assignment, 2)))  
else:  
    print("unsat")
```




```
from boolector import Boolector
```

```
b = Boolector()
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

Print Solution

```
if res == b.SAT:
```

```
    print("{} {}".format(x.symbol, int(x.assignment, 2)))
```

```
else:
```

```
    print("unsat")
```



```
from boolector import Boolector
```

```
b = Boolector()
b.Set_opt("model_gen", 1)
```

```
const = b.Const(133713378, 64)
x      = b.Var(64, "x")
```

```
b.Assert(x*20 == 100)
b.Assert(b.Ult(x, 100))
```

```
res = b.Sat()
```

```
if res == b.SAT:
    print("{} {}".format(x.symbol, int(x.assignment, 2)))
else:
    print("unsat")
```

```
:- (
```



TASK



Find x (int64_t) such that:

$$x < 0$$

$$x * 133713378 == 13372$$



Task 1

Find x (int64_t) such that:

$$x < 0$$

$$x * 133713378 == 13372$$

code in: ~/smt/task_1.py

`b.Slt(x,y)` //Signed less than

Hint: There is a solution

Tips



Universality



So far: "Find x, y, z such that ψ becomes true"



Variables

So far: "Find x, y, z such that ψ becomes true"



Any formula

So far: "Find x, y, z such that ψ becomes true"



Now: "is ψ always true?"



Now: "is ψ always true?"



not(ψ) has no solution



Now: "is ψ always true?"



not(ψ) has no solution



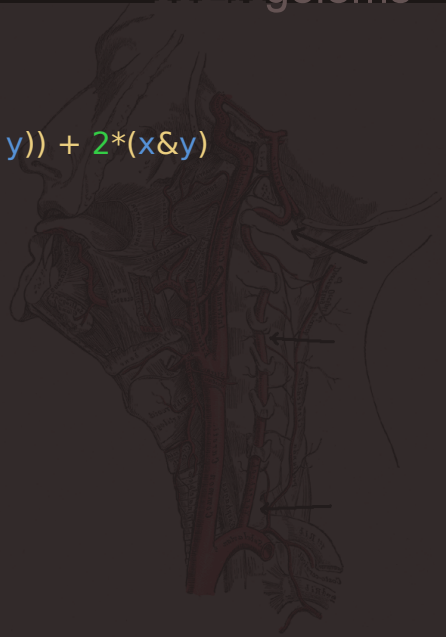
ask Solver!



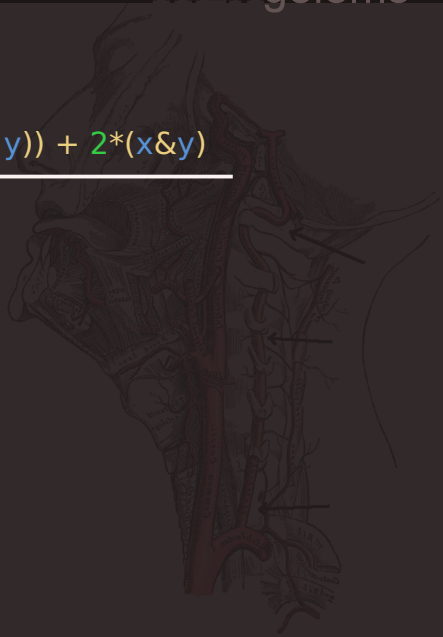
TASK



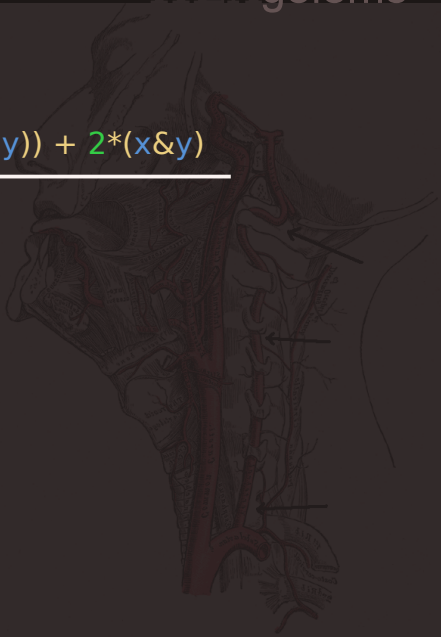
$$((x \mid y) \& \sim(x \& y)) + 2^*(x\&y)$$



$$x \quad y \quad ((x | y) \& \sim(x \& y)) + 2^*(x \& y)$$



x	y	$((x \mid y) \& \sim(x \& y)) + 2*(x\&y)$
1	1	2



x	y	$((x \mid y) \& \sim(x \& y)) + 2*(x\&y)$
1	1	2
5	1	6



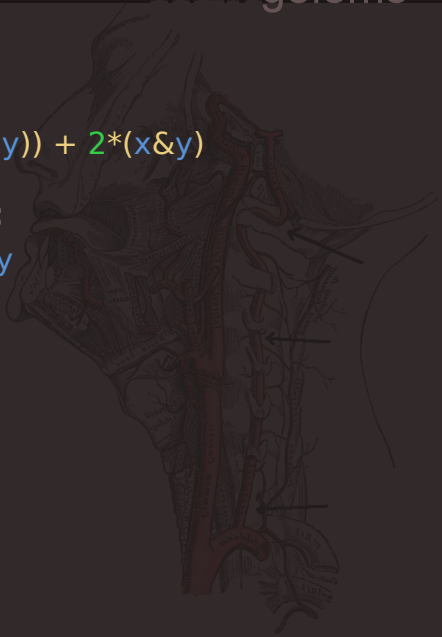
x	y	$((x \mid y) \& \sim(x \& y)) + 2*(x\&y)$
1	1	2
5	1	6
2	5	7



$$((x \mid y) \& \sim(x \& y)) + 2*(x\&y)$$

?

$$x+y$$



Task 2

Are

$$((x \mid y) \& \sim(x \& y)) + 2*(x\&y)$$

and

$$x+y$$

always the same?



Task 2

Are

$((x \mid y) \& \sim(x \& y)) + 2*(x\&y)$

and

$x+y$

always the same?

Tips

code in: `~/smt/task_2.py`

`~x` //Binary negation



Bonus Task

```
if((x*x+x)%2 == 0){  
    //[...]  
}else{  
    //[...]  
}
```



Bonus Task

WTF?

```
if((x*x+x)%2 == 0){  
    //[...]  
}else{  
    //[...]  
}
```



Bonus Task

```
if((x*x+x)%2 == 0){  
    //[...] ←  
}else{  
    //[...] ←  
}
```

Can we get here?



Bonus Task

```
if((x*x+x)%2 == 0){  
    //[...] ←  
}else{  
    //[...] ←  
}
```

Can we get here?

(Yes $x = 0$)



Bonus Task

```
if((x*x+x)%2 == 0){  
    //[...]  
}else{  
    //[...]←  
}
```

Can we get here?



Workflow



Workflow

?



```
x < 5  
x + 3 >= 4  
x * 2 < 3  
x > 0
```

Translate



Workflow



$x < 5$
 $x + 3 \geq 4$
 $x * 2 < 3$
 $x > 0$



$x = 1$

Translate

SMT



INPUT CRAFTING



$$y = P(x)$$



$$y = P(x)$$



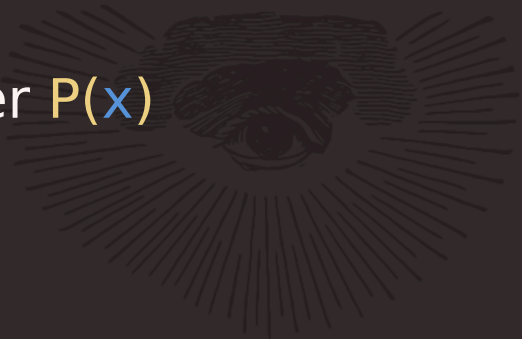
Inputs



$$y = P(x)$$



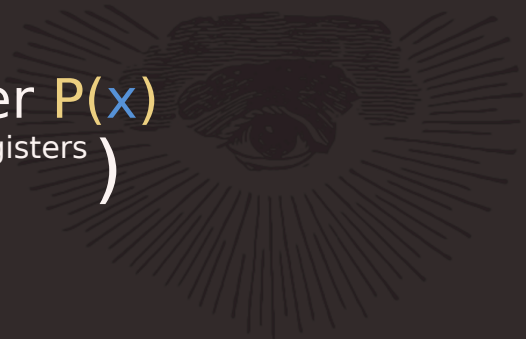
State after $P(x)$



$$y = P(x)$$



State after $P(x)$
(Variables / Registers)
Memory



$$y = P(x)$$

Is there x such that $\psi(y)$?



$$y = P(x)$$

Is there x such that $\psi(y)$?

Translate

(Formula that "runs" P) $\&\& \psi(y)$



$$y = P(x)$$

Is there x such that $\psi(y)$?

Translate

(Formula that "runs" P) $\&\& \psi(y)$



Solver finds x



```
uint64_t x = read_uint();  
uint64_t a = x ^ 0xd701ecf9bd67d788;  
uint64_t b = x * 0x94d941135c908617;  
uint64_t c = a + b;
```



```
uint64_t x = read_uint();  
uint64_t a = x ^ 0xd701ecf9bd67d788;  
uint64_t b = x * 0x94d941135c908617;  
uint64_t c = a + b;  
// c == 0x1
```



Translate

```
uint64_t x = read_uint();  
uint64_t a = x ^ 0xd701ecf9bd67d788;  
uint64_t b = x * 0x94d941135c908617;  
uint64_t c = a + b;  
// c == 0x1
```



Translate

```
uint64_t x = read_uint();  
uint64_t a = x ^ 0xd701ecf9bd67d788;  
uint64_t b = x * 0x94d941135c908617;  
uint64_t c = a + b;  
Assert(c == 0x1)
```




Translate

```
uint64_t x = read_uint();  
uint64_t a = x ^ 0xd701ecf9bd67d788;  
uint64_t b = x * 0x94d941135c908617;  
Assert(c == a + b)  
Assert(c == 0x1)
```



Translate

```
uint64_t x = read_uint();  
uint64_t a = x ^ 0xd701ecf9bd67d788;  
uint64_t b = x * 0x94d941135c908617;  
Assert(c == a + b)  
Assert(c == 0x1)
```



Assert, not Assign



Translate

```
uint64_t x = read_uint();  
uint64_t a = x ^ 0xd701ecf9bd67d788;  
Assert(b == x * 0x94d941135c908617)  
Assert(c == a + b)  
Assert(c == 0x1)
```



Translate

```
uint64_t x = read_uint();  
Assert(a == x ^ 0xd701ecf9bd67d788)  
Assert(b == x * 0x94d941135c908617)  
Assert(c == a + b)  
Assert(c == 0x1)
```




Translate

```
x = Var(64, "x")  
Assert(a == x ^ 0xd701ecf9bd67d788)  
Assert(b == x * 0x94d941135c908617)  
Assert(c == a + b)  
Assert(c == 0x1)
```

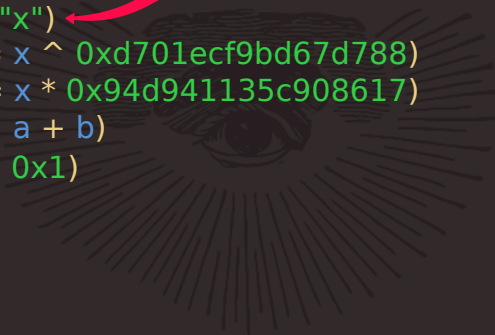


Translate

Craft Input



```
x = Var(64, "x")  
Assert(a == x ^ 0xd701ecf9bd67d788)  
Assert(b == x * 0x94d941135c908617)  
Assert(c == a + b)  
Assert(c == 0x1)
```



Translate

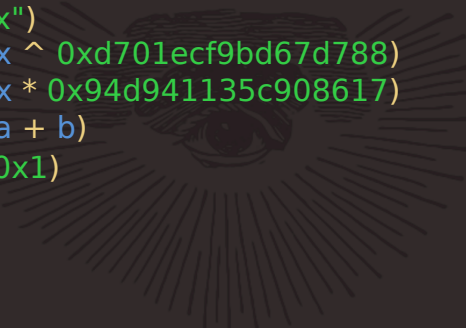
```
a = Var(64, "a")  
b = Var(64, "b")  
c = Var(64, "c")  
x = Var(64, "x")  
Assert(a == x ^ 0xd701ecf9bd67d788)  
Assert(b == x * 0x94d941135c908617)  
Assert(c == a + b)  
Assert(c == 0x1)
```



Translate

Necessary Evil

```
a = Var(64, "a")  
b = Var(64, "b")  
c = Var(64, "c")  
x = Var(64, "x")  
Assert(a == x ^ 0xd701ecf9bd67d788)  
Assert(b == x * 0x94d941135c908617)  
Assert(c == a + b)  
Assert(c == 0x1)
```



```
x = x + 1  
a = x + 2
```



Translate

```
x = x + 1
```

```
a = x + 2
```



```
Assert(x == x + 1)
```

```
Assert(a == x + 2)
```

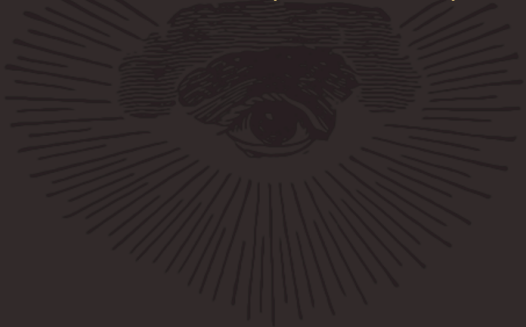


Unsat

Translate

```
x = x + 1  
a = x + 2
```

```
Assert(x == x + 1)  
Assert(a == x + 2)
```

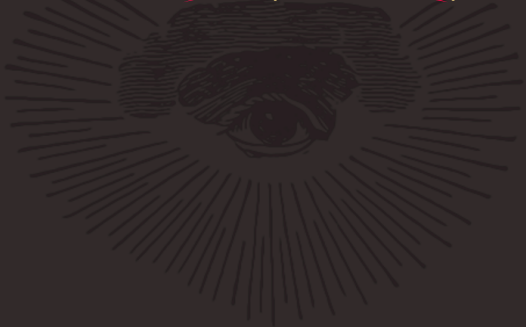


Unsat

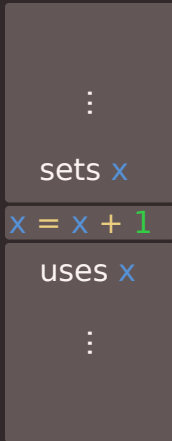
Translate

```
x = x + 1  
a = x + 2
```

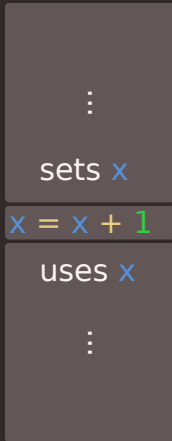
```
Assert(x == x + 1)  
Assert(a == x + 2)
```




SSA



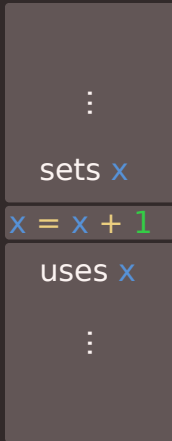
SSA



code where x is
assigned



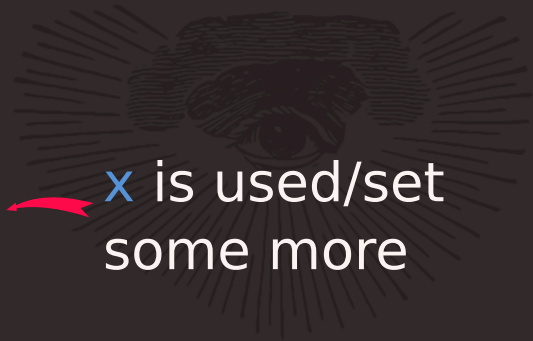
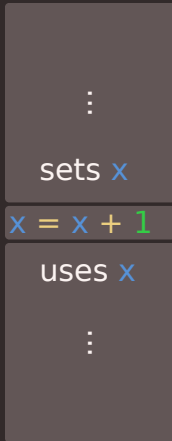
SSA



x assigned
second time



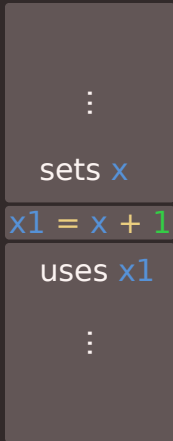
SSA



x is used/set
some more



SSA



Translate

```
x = x + 1  
a = x + 2
```



```
Assert(x == x + 1)  
Assert(a == x + 2)
```



Translate

```
x = x + 1
```

```
a = x + 2
```

```
Assert(x == x + 1)
```

```
Assert(a == x + 2)
```

SSA

```
x1 = x + 1
```

```
a = x1 + 2
```



Translate

```
x = x + 1  
a = x + 2
```

SSA

```
x1 = x  + 1  
a = x1 + 2
```

```
Assert(x == x + 1)  
Assert(a == x + 2)
```

```
Assert(x1 == x + 1)  
Assert(a == x1 + 2)
```





Task 3

```
int main(){  
    uint64_t x = read_uint();  
    uint64_t a = 0;  
  
    a += x;  
    x = x ^ 0xd701ecf9bd67d788;  
    a += x;  
    x = x * 0x94d941135c908617;  
    a += x;  
    return a;  
}
```

find input **x**

xor

to return **1**



CONTROL FLOW




```
if(i == 0) {  
    x = x + 5;  
} else {  
    x = x + 3;  
}  
y = x;
```



if(i == 0) {
 x = x + 5;
} **else** {
 x = x + 3;
}
y = x;

SSA

if(i == 0) {
 x1 = x + 5;
} **else** {
 x2 = x + 3;
}
y = x???



```
if(i == 0) {  
    x = x + 5;  
} else {  
    x = x + 3;  
}  
y = x;
```

SSA



```
if(i == 0) {  
    x1 = x + 5;  
} else {  
    x2 = x + 3;  
}  
y = (i==0)? x1 : x2;
```



```
if(i == 0) {  
    x = x + 5;  
} else {  
    x = x + 3;  
}  
y = x;
```

SSA



```
x1 = x + 5;  
x2 = x + 3;  
y = (i == 0)? x1 : x2;
```



```
if(i == 0) {  
    x = x + 5;  
} else {  
    x = x + 3;  
}  
y = x;
```

SSA

→

```
x1 = x + 5;  
x2 = x + 3;  
y = (i==0)? x1 : x2;
```

↙ Translate

```
Assert(x1 == x + 5)  
Assert(x2 == x + 3)  
Assert(y == Cond( i==0, x1, x2 ) )
```



TASK



Task 4

Translate:

```
int32_t v = read_int();  
char c = read_char();  
if( c == '-' ){  
    v-=1;  
else{  
    v*=2;  
}  
// v == -1 && c > 'A'
```



Loops

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```



Loops

```
int i = 0;
```

Unroll

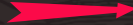
```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```



Loops

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



```
int i = 0;  
Assert( i<n );  
body(i);  
i++;
```



Loops

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



```
int i = 0;  
Assert( i<n );  
body(i);  
i++;  
Assert( i<n );  
body(i);  
i++;
```



Loops

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



```
int i = 0;  
Assert( i<n );  
body(i);  
i++;  
Assert( i<n );  
body(i);  
i++;  
[...]
```



Loops

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



```
int i = 0;  
Assert( i<n );  
body(i);  
i++;  
Assert( i<n );  
body(i);  
i++;  
[...]  
Assert( !(i<n) );
```



Exact number of Iterations needed



Exact number
of Iterations needed

BAD

(but for now we can live with it)



TASK



Task 5

Translate:

```
uint32_t v = read_uint();  
uint32_t r = v;  
for(int i = 0; i < 32; i++){  
    r = r ^ (v << i) * (v + r);  
}  
// r == 2016
```



Task 5

Translate:

```
uint32_t v = read_uint();  
uint32_t r = v;  
for(int i = 0; i < 32; i++){  
    r = r ^ (v << i) * (v + r);  
}  
// r == 2016
```

Tips

Unroll:

```
for i in range(32):  
    r_next = Var(32, "r"+str(i))  
    Assert( r_next == r ^ ... )  
    r = r_next
```



TASK



Task 6

Translate:

```
int check( char* str, int len ){
    uint64_t v = 0;
    for( int i = 0; i < len; i++ ){
        if( str[i] == '-' ){
            v -= 1;
        } else {
            v *= 2;
        }
    }
    return v == 0xffffffffbadc0de0;
}
```



CRYPTO



Task 7

```
def hash_func(inputstr):  
    crc = 0  
    for bit in inputstr:  
        if int(bit) == most_significant_bit(crc):  
            crc = (crc << 1) & 0xffffffff  
        else:  
            crc = ((crc << 1) ^ 0x04C11DB7) & 0xffffffff  
    return crc
```

```
def bitstring(str): [...]
```



Task 7

```
def hash_func(inputstr): "100010001"  
    crc = 0  
    for bit in inputstr:  
        if int(bit) == most_significant_bit(crc):  
            crc = (crc << 1) & 0xffffffff  
        else:  
            crc = ((crc << 1) ^ 0x04C11DB7) & 0xffffffff  
    return crc
```

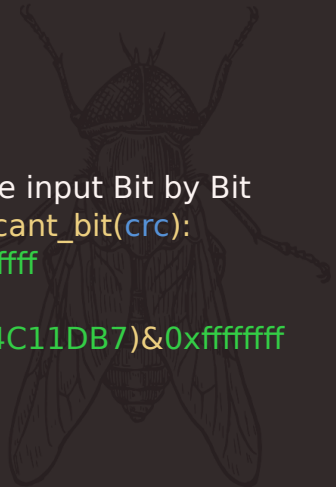
```
def bitstring(str): [...]
```



Task 7

```
def hash_func(inputstr):  
    crc = 0  
    for bit in inputstr:      Iterate input Bit by Bit  
        if int(bit) == most_significant_bit(crc):  
            crc = (crc << 1) & 0xffffffff  
        else:  
            crc = ((crc << 1) ^ 0x04C11DB7) & 0xffffffff  
    return crc
```

```
def bitstring(str): [...]
```



Task 7

```
def hash_func(inputstr):  
    crc = 0  
    for bit in inputstr:  
        if int(bit) == most_significant_bit(crc):  
            crc = (crc << 1) & 0xffffffff  
        else:  
            crc = ((crc << 1) ^ 0x04C11DB7) & 0xffffffff  
    return crc
```

Update crc based on bit

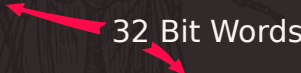
```
def bitstring(str): [...]
```



Task 7

```
def hash_func(inputstr):  
    crc = 0  
    for bit in inputstr:  
        if int(bit) == most_significant_bit(crc):  
            crc = (crc << 1) & 0xffffffff  
        else:  
            crc = ((crc << 1) ^ 0x04C11DB7) & 0xffffffff  
    return crc
```

32 Bit Words



```
def bitstring(str): [...]
```



Task 7

```
def hash_func(inputstr):  
    crc = 0  
    for bit in inputstr:  
        if int(bit) == most_significant_bit(crc):  
            crc = (crc << 1) & 0xffffffff  
        else:  
            crc = ((crc << 1) ^ 0x04C11DB7) & 0xffffffff  
    return crc
```

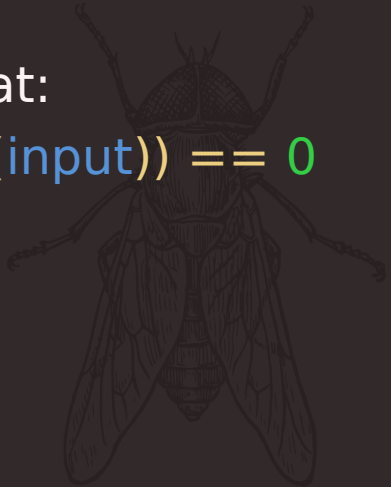
```
def bitstring(str): [...] String to Bit-String
```



Task 7

Find `input` such that:

- `hash(bitstring(input))) == 0`



Task 7

Find `input` such that:

- `hash(bitstring(input)) == 0`
- `input` contains only "+-<>;_"



~~DIY~~



Active Python Runs on Binary



github.com/angr



Miasm

Active
Python
Runs on Binary

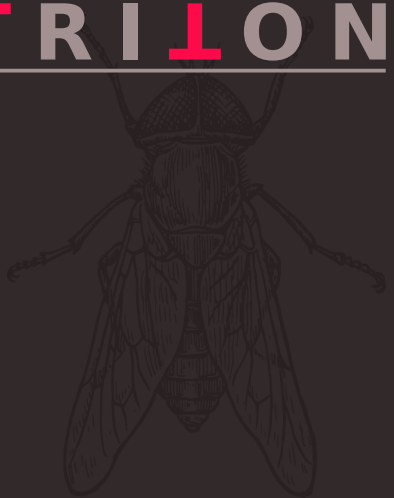


github.com/cea-sec/miasm



TRILON

Active
C++/Python
Runs on Binary



triton.quarkslab.com



BAP

Active
Ocaml/(Python)
Runs on Binary



github.com/BinaryAnalysisPlatform/bap



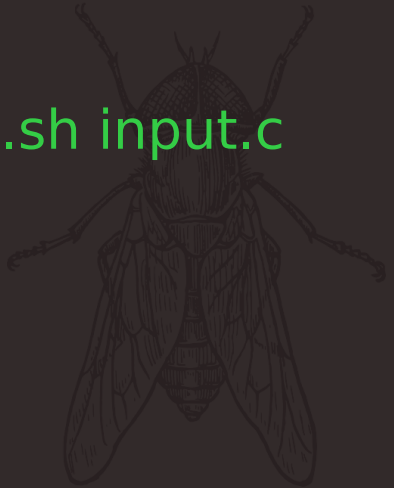
Inactive
Not OS
Runs on C
Standalone



[llbmc.org](http://lbmc.org)



```
$ sh run_llbmc.sh input.c
```



```
$ sh run_llbmc.sh input.c
```

(add parameters to llbmc in run_llbmc.sh)

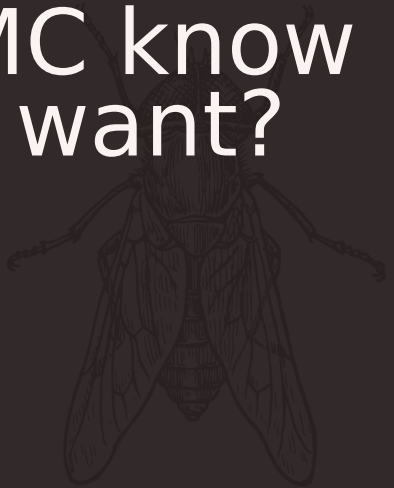


```
$ sh run_llbmc.sh input.c
```

```
#include "llbmc.h"
```



Does LLBMC know what we want?

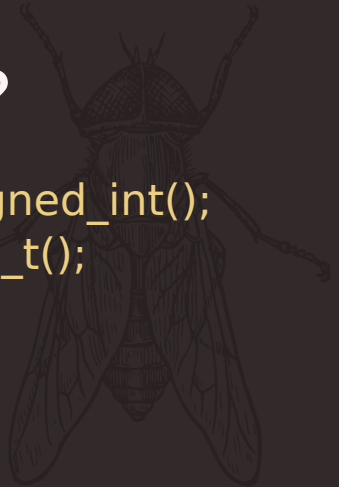


Define Inputs?



Define Inputs?

```
__llbmc_nondef_unsigned_int();  
__llbmc_nondef_uint8_t();
```



Define Inputs?

et al.

```
__llbmc_nondef_unsigned_int();  
__llbmc_nondef_uint8_t();
```

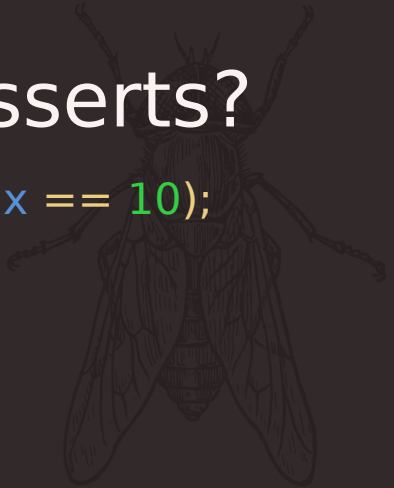


Additional Asserts?



Additional Asserts?

```
__lbmc_assert(q->x == 10);
```

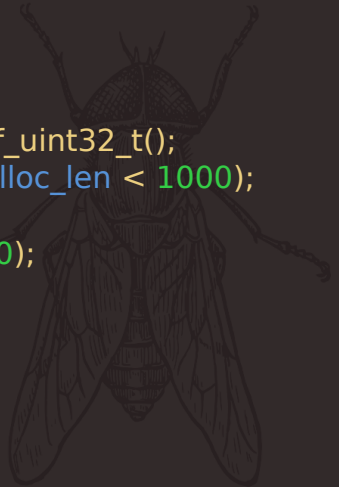


Additional Asserts?

```
__llbmc_assert(q->x == 10);  
__llbmc_assume(0 < s);
```



```
int main(){
    uint32_t alloc_len = __llbmc_nondef_uint32_t();
    __llbmc_assume(alloc_len > 0 && alloc_len < 1000);
    char* str = malloc(alloc_len);
    __llbmc_assume(str[alloc_len-1]==0);
    size_t len = strlen(str);
    __llbmc_assert(len < alloc_len);
    return 0;
}
```



```
int main(){
    uint32_t alloc_len = __llbmc_nondef_uint32_t();
    __llbmc_assume(alloc_len > 0 && alloc_len < 1000);
    char* str = malloc(alloc_len);
    __llbmc_assume(str[alloc_len-1]==0);
    size_t len = strlen(str);    Run function
    __llbmc_assert(len < alloc_len);
    return 0;
}
```



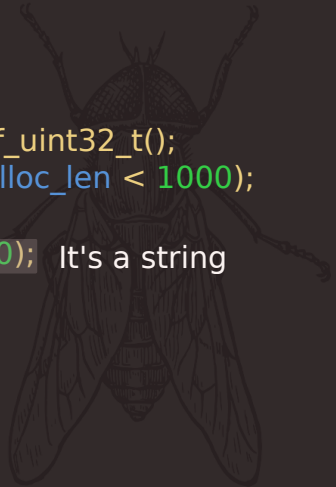
Test Harness

```
int main(){  
    uint32_t alloc_len = __llbmc_nondef_uint32_t();  
    __llbmc_assume(alloc_len > 0 && alloc_len < 1000);  
    char* str = malloc(alloc_len);  
    __llbmc_assume(str[alloc_len-1]==0);  
    size_t len = strlen(str);  
    __llbmc_assert(len < alloc_len);  
    return 0;  
}
```

Create allocation




```
int main(){
    uint32_t alloc_len = __llbmc_nondef_uint32_t();
    __llbmc_assume(alloc_len > 0 && alloc_len < 1000);
    char* str = malloc(alloc_len);
    __llbmc_assume(str[alloc_len-1]==0); It's a string
    size_t len = strlen(str);
    __llbmc_assert(len < alloc_len);
    return 0;
}
```



```
int main(){
    uint32_t alloc_len = __llbmc_nondef_uint32_t();
    __llbmc_assume(alloc_len > 0 && alloc_len < 1000);
    char* str = malloc(alloc_len);
    __llbmc_assume(str[alloc_len-1]==0);
    size_t len = strlen(str);
    __llbmc_assert(len < alloc_len);
    return 0;
}
```

Check some property



TASK



Task 8

Use LLBMC to solve Task 4
in `~/smt/task_8.c`



DEMO

(Prayer to the demo gods)



TASK



Task 9

Use LLBMC to find bugs
in `~/smt/task_9.c`



```
#include "llbmc.h"
```

Define Inputs?

```
__llbmc_nondef_unsigned_int();  
__llbmc_nondef_uint8_t();
```

Additional Asserts?

```
__llbmc_assert(q->x == 10);  
__llbmc_assume(0 < s);
```

```
$ sh run_llbmc.sh input.c
```



Thanks



Loops (cont'd)

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```



Loops (cont'd)

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



```
int i = 0;  
Assert( i<n );  
body(i);  
i++;  
Assert( i<n );  
body(i);  
i++;  
[...]  
Assert( !(i<n) );
```



Loops (cont'd)

Exact number
of Iterations needed

BAD



Loops (cont'd)

```
int i = 0;
```

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



Loops (cont'd)

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll

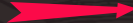
```
int i = 0;  
if(i<n){  
    body(i);  
    i++;  
}
```



Loops (cont'd)

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



```
int i = 0;  
if(i<n){  
    body(i);  
    i++;  
if( i<n ){  
    body(i);  
    i++;  
}  
}
```



Loops (cont'd)

```
for( int i = 0; i<n; i++ ){  
    body(i);  
}
```

Unroll



```
int i = 0;  
if(i<n){  
    body(i);  
    i++;  
if( i<n ){  
    body(i);  
    i++;  
}  
}  
Assert( !(i<n) );
```

