CCS'25

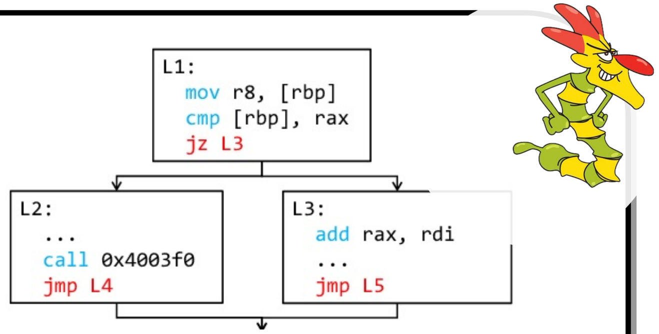
Augmenting Search-based Program Synthesis with Local Inference Rules to Improve Black-box Deobfuscation

Vidal Attias (CEA LIST, Loria)

Nicolas Bellec & **Grégoire Menguy** & Sébastien Bardin (CEA LIST)

Jean-Yves Marion (Loria)

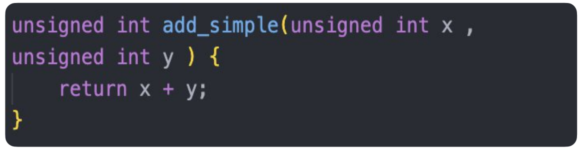
Obfuscation and Deobfuscation



```
L1:
  mov r8, [rbp]
  cmp [rbp], rax
  jz L3

L2:
  ...
  call 0x4003f0
  jmp L4

L3:
  add rax, rdi
  ...
  jmp L5
```



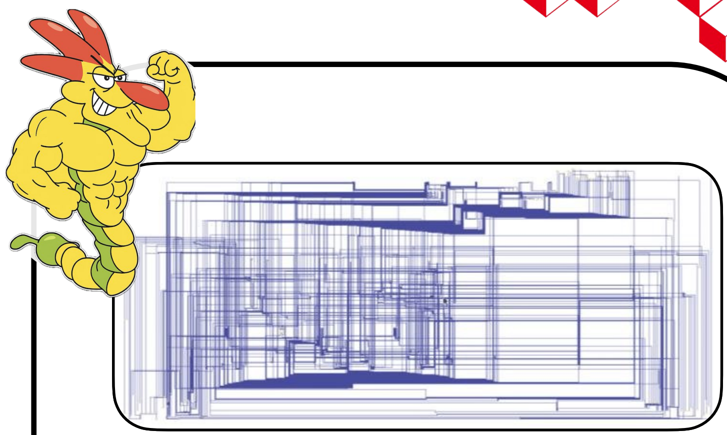
```
unsigned int add_simple(unsigned int x ,
unsigned int y ) {
    return x + y;
}
```

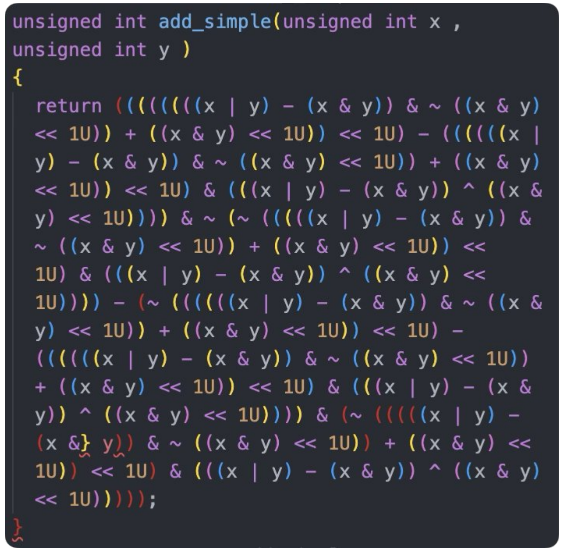
Obfuscation

☣ Thwart analysis and detection

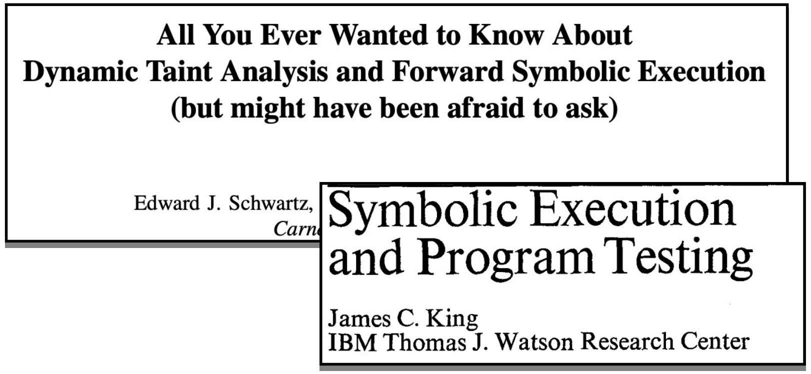
Deobfuscation

☣ Help malware analysis





```
unsigned int add_simple(unsigned int x ,
unsigned int y )
{
  return ((((((((x | y) - (x & y)) & ~ ((x & y)
  << 1U)) + ((x & y) << 1U)) << 1U) - ((((((x |
  y) - (x & y)) & ~ ((x & y) << 1U)) + ((x & y)
  << 1U)) << 1U) & (((x | y) - (x & y)) ^ ((x &
  y) << 1U)))) & ~ (~ (((((x | y) - (x & y)) &
  ~ ((x & y) << 1U)) + ((x & y) << 1U)) <<
  1U) & (((x | y) - (x & y)) ^ ((x & y) <<
  1U)))) - (~ ((((((x | y) - (x & y)) & ~ ((x &
  y) << 1U)) + ((x & y) << 1U)) << 1U) -
  ((((((x | y) - (x & y)) & ~ ((x & y) << 1U))
  + ((x & y) << 1U)) << 1U) & (((x | y) - (x &
  y)) ^ ((x & y) << 1U)))) & (~ (((((x | y) -
  (x &} y)) & ~ ((x & y) << 1U)) + ((x & y) <<
  1U)) << 1U) & (((x | y) - (x & y)) ^ ((x & y)
  << 1U))));
}
```



All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)

Edward J. Schwartz,
Carn

Symbolic Execution and Program Testing

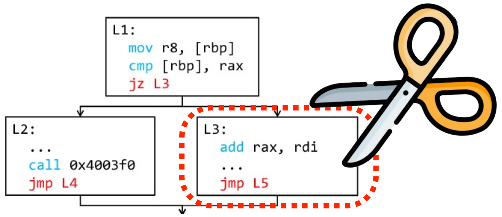
James C. King
IBM Thomas J. Watson Research Center

BINSEC

Problem: anti-white-box deobfuscation

Breakthrough 2017: Black-Box Deobfuscation

1 Defining a reverse window



```
L1:
  mov r8, [rbp]
  cmp [rbp], rax
  jz L3

L2:
  ...
  call 0x4003f0
  jmp L4

L3:
  add rax, rdi
  ...
  jmp L5
```

Obfuscated binary

```
L3:
  add rax, rdi
  ...
  jmp L5
```

Region of interest

2 Sampling I/O

x = 0xD142
x = 0x7A0C
x = 0x0ABE
x = 0xE8B3

Obfuscated code

f(x) = 0xD144
f(x) = 0x7A0E
f(x) = 0x0AC0
f(x) = 0xE8B5

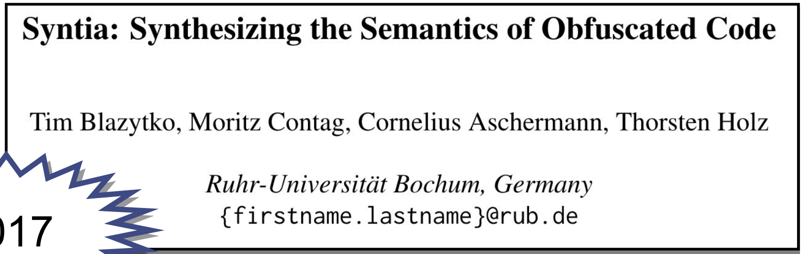
3 Synthesizing simpler expression

x	f(x)
0xD142 →	0xD144
0x7A0C →	0x7A0E
0x0ABE →	0x0AC0
0xE8B3 →	0xE8B5

SyGuS
Syntax-Guided Synthesis

Synthesizer

f(x) = x + 2

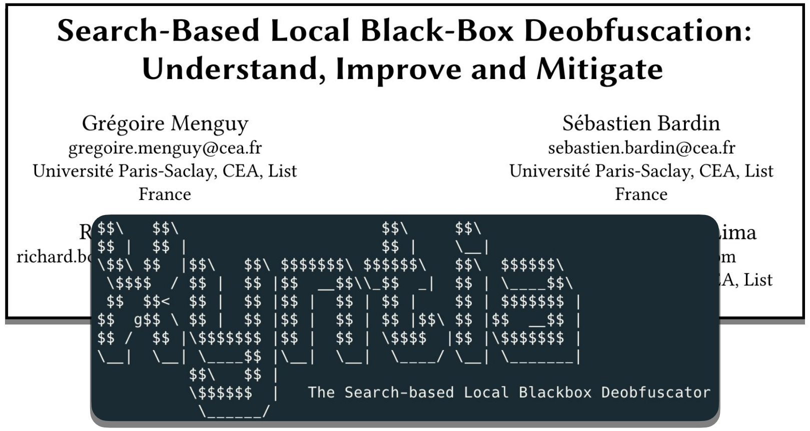


Syntia: Synthesizing the Semantics of Obfuscated Code

Tim Blazytko, Moritz Contag, Cornelius Aschermann, Thorsten Holz

Ruhr-Universität Bochum, Germany
{firstname.lastname}@rub.de

2017



Search-Based Local Black-Box Deobfuscation: Understand, Improve and Mitigate

Grégoire Menguy
gregoire.menguy@cea.fr
Université Paris-Saclay, CEA, List
France

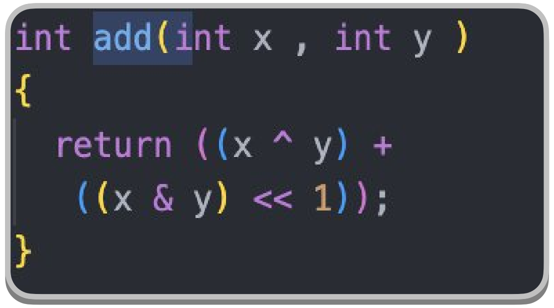
Sébastien Bardin
sebastien.bardin@cea.fr
Université Paris-Saclay, CEA, List
France

R
richard.b

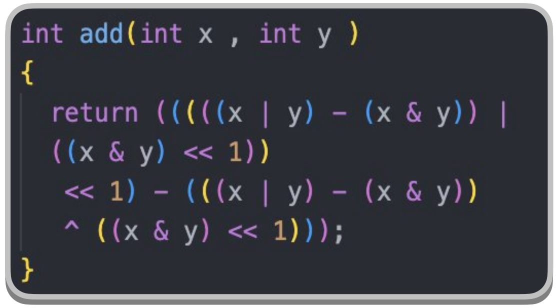
ima
m
A, List

```
$$\   $$\                         $$\     $$\
$$ |  $$ |                        $$ |    \__|
\$$\ $$  |$$\   $$\ $$$$$$$\ $$$$$$\   $$\  $$$$$$\
 \$$$$  / $$ |  $$ |$$  __$$\\_$$  _|  $$ | \____$$\
 $$  $$<  $$ |  $$ |$$ |  $$ | $$ |    $$ | $$$$$$$ |
$$  g$$ \ $$ |  $$ |$$ |  $$ | $$ |$$\ $$ |$$  __$$ |
$$ /  $$ |\$$$$$$$ |$$ |  $$ | \$$$$  |$$ |\$$$$$$$ |
\__|  \__| \____$$ |\__|  \__|  \____/ \__| \_______|
          $$\   $$ |
          \$$$$$$  |   The Search-based Local Blackbox Deobfuscator
           \______/
```

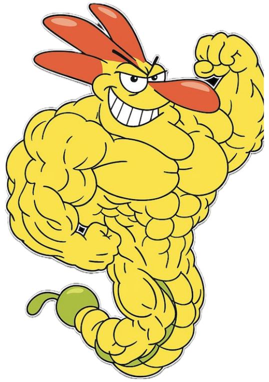
Black-Box is Immune to Syntactic Complexity

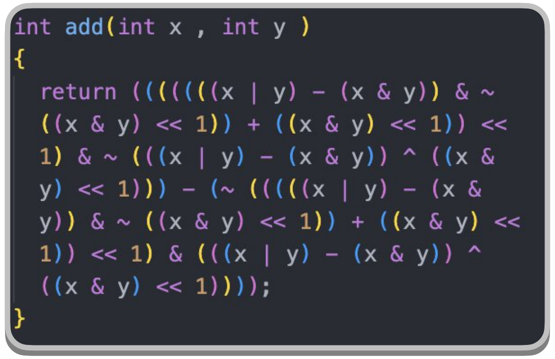


```
int add(int x , int y )
{
  return ((x ^ y) +
   ((x & y) << 1));
}
```



```
int add(int x , int y )
{
  return (((((x | y) - (x & y)) |
  ((x & y) << 1))
   << 1) - (((x | y) - (x & y))
   ^ ((x & y) << 1)));
}
```





```
int add(int x , int y )
{
  return (((((((x | y) - (x & y)) & ~
  ((x & y) << 1)) + ((x & y) << 1)) <<
  1) & ~ (((x | y) - (x & y)) ^ ((x &
  y) << 1))) - (~ (((((x | y) - (x &
  y)) & ~ ((x & y) << 1)) + ((x & y) <<
  1)) << 1) & (((x | y) - (x & y)) ^
  ((x & y) << 1))));
}
```

White Box

😎

🥵

☠

Black Box

😎

😎

😎

Under the Hood

Black-box deobfuscation is based on *program synthesis over BV theory*

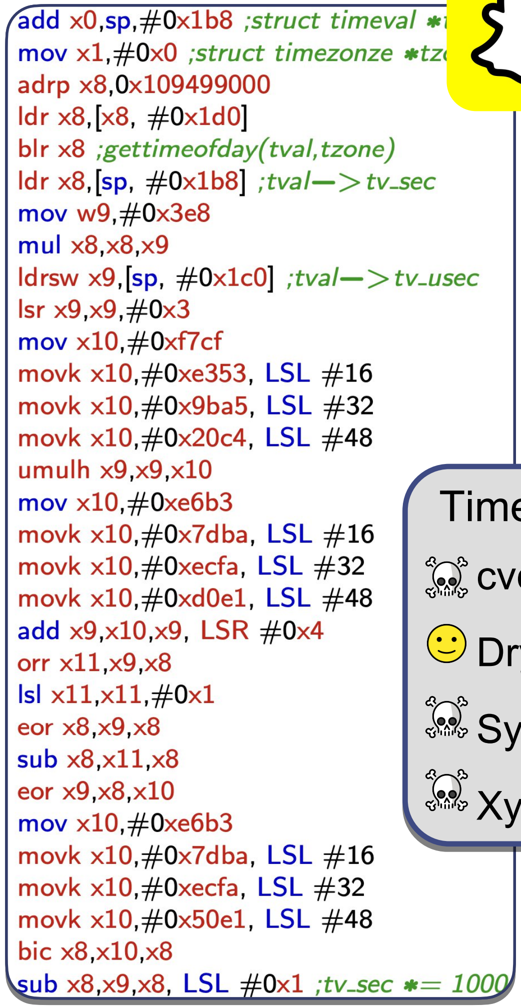
1. **Dedicated deobfuscation synthesizers**
   - Syntia
   - Xyntia
2. **General purpose synthesizers**
   - CVC4 / cvc5
   - DryadSynth

But there are limitations

- Constant values
- Large expressions

⇨ **Black-box essentially for VM-handlers**

Real example from Snapchat iOS app



```
add x0,sp,#0x1b8 ;struct timeval *t
mov x1,#0x0 ;struct timezonze *tz
adrp x8,0x109499000
ldr x8,[x8, #0x1d0]
blr x8 ;gettimeofday(tval,tzone)
ldr x8,[sp, #0x1b8] ;tval—>tv_sec
mov w9,#0x3e8
mul x8,x8,x9
ldrsw x9,[sp, #0x1c0] ;tval—>tv_usec
lsr x9,x9,#0x3
mov x10,#0xf7cf
movk x10,#0xe353, LSL #16
movk x10,#0x9ba5, LSL #32
movk x10,#0x20c4, LSL #48
umulh x9,x9,x10
mov x10,#0xe6b3
movk x10,#0x7dba, LSL #16
movk x10,#0xecfa, LSL #32
movk x10,#0xd0e1, LSL #48
add x9,x10,x9, LSR #0x4
orr x11,x9,x8
lsl x11,x11,#0x1
eor x8,x9,x8
sub x8,x11,x8
eor x9,x8,x10
mov x10,#0xe6b3
movk x10,#0x7dba, LSL #16
movk x10,#0xecfa, LSL #32
movk x10,#0x50e1, LSL #48
bic x8,x10,x8
sub x8,x9,x8, LSL #0x1 ;tv_sec *= 1000
```

Timeout: 1h

☠ cvc5

🙂 DryadSynth

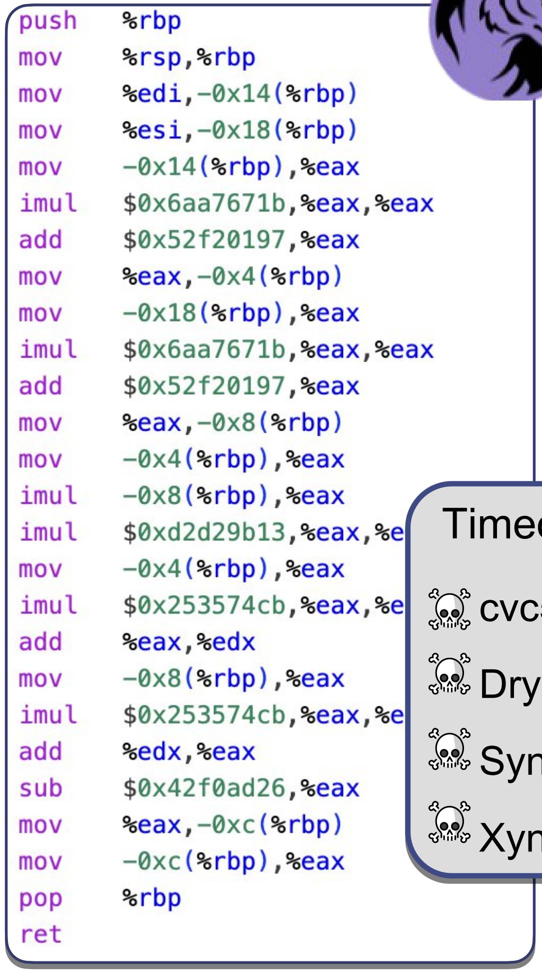
☠ Syntia

☠ Xyntia

Expression:

y = x * 1000

Tigress example



```
push    %rbp
mov     %rsp,%rbp
mov     %edi,-0x14(%rbp)
mov     %esi,-0x18(%rbp)
mov     -0x14(%rbp),%eax
imul    $0x6aa7671b,%eax,%eax
add     $0x52f20197,%eax
mov     %eax,-0x4(%rbp)
mov     -0x18(%rbp),%eax
imul    $0x6aa7671b,%eax,%eax
add     $0x52f20197,%eax
mov     %eax,-0x8(%rbp)
mov     -0x4(%rbp),%eax
imul    -0x8(%rbp),%eax
imul    $0xd2d29b13,%eax,%e
mov     -0x4(%rbp),%eax
imul    $0x253574cb,%eax,%e
add     %eax,%edx
mov     -0x8(%rbp),%eax
imul    $0x253574cb,%eax,%e
add     %edx,%eax
sub     $0x42f0ad26,%eax
mov     %eax,-0xc(%rbp)
mov     -0xc(%rbp),%eax
pop     %rbp
ret
```

Timeout: 1h

☠ cvc5

☠ DryadSynth

☠ Syntia

☠ Xyntia

Expression:

z = x * y * 0x6aa7671b + 0x52f20197

Our Contribution

Black-box:
From VM-handlers to
arbitrary code blocks

1 Search Modulo Inference Rules (SMIR)

↳ Combine **search-based synthesis** with **symbolic reasoning**

- Automatically *elevate* candidate solutions
- Guide the search

2 XSMIR : Implementation of SMIR on top of Xyntia

↳ Artifacts : zenodo Artifacts Available V1.1 Artifacts Evaluated Functional V1.1 Results Reproduced V1.1

xyntia Public

https://github.com/binsec/xyntia

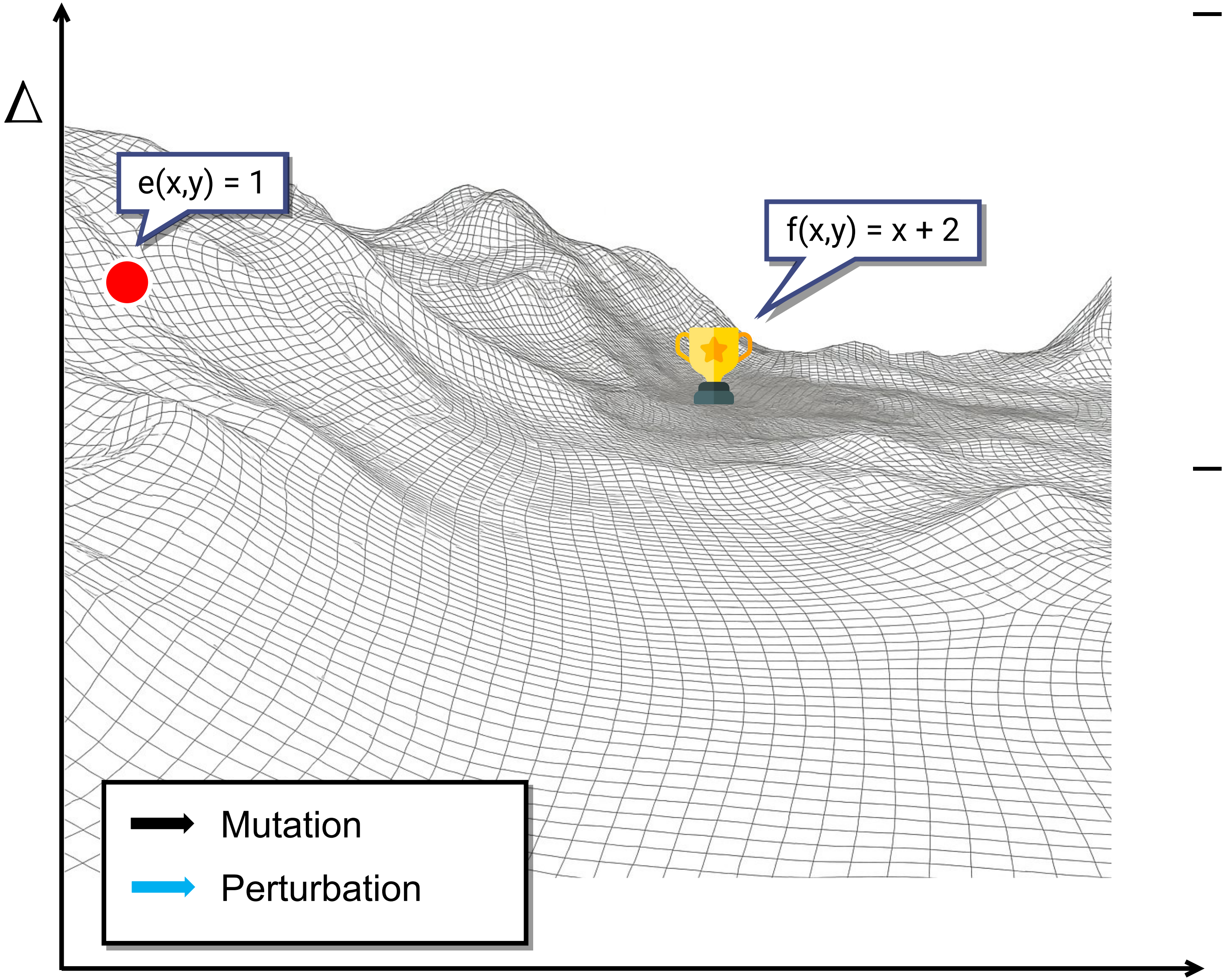
3 First evaluation of black-box deobfuscation **at scale on binary-level code blocks**

↳ Real binaries (obfuscated, malware, in-the-wild), MBA and synthetic expressions

↳ XSMIR outperforms **black-box deobfuscators**, **PL synthesizers** and **MBA deobfuscators**

(Syntia, Xyntia) (CVS4/5, DryadSynth) (ProMBA, GAMBA)

Search-based Synthesis



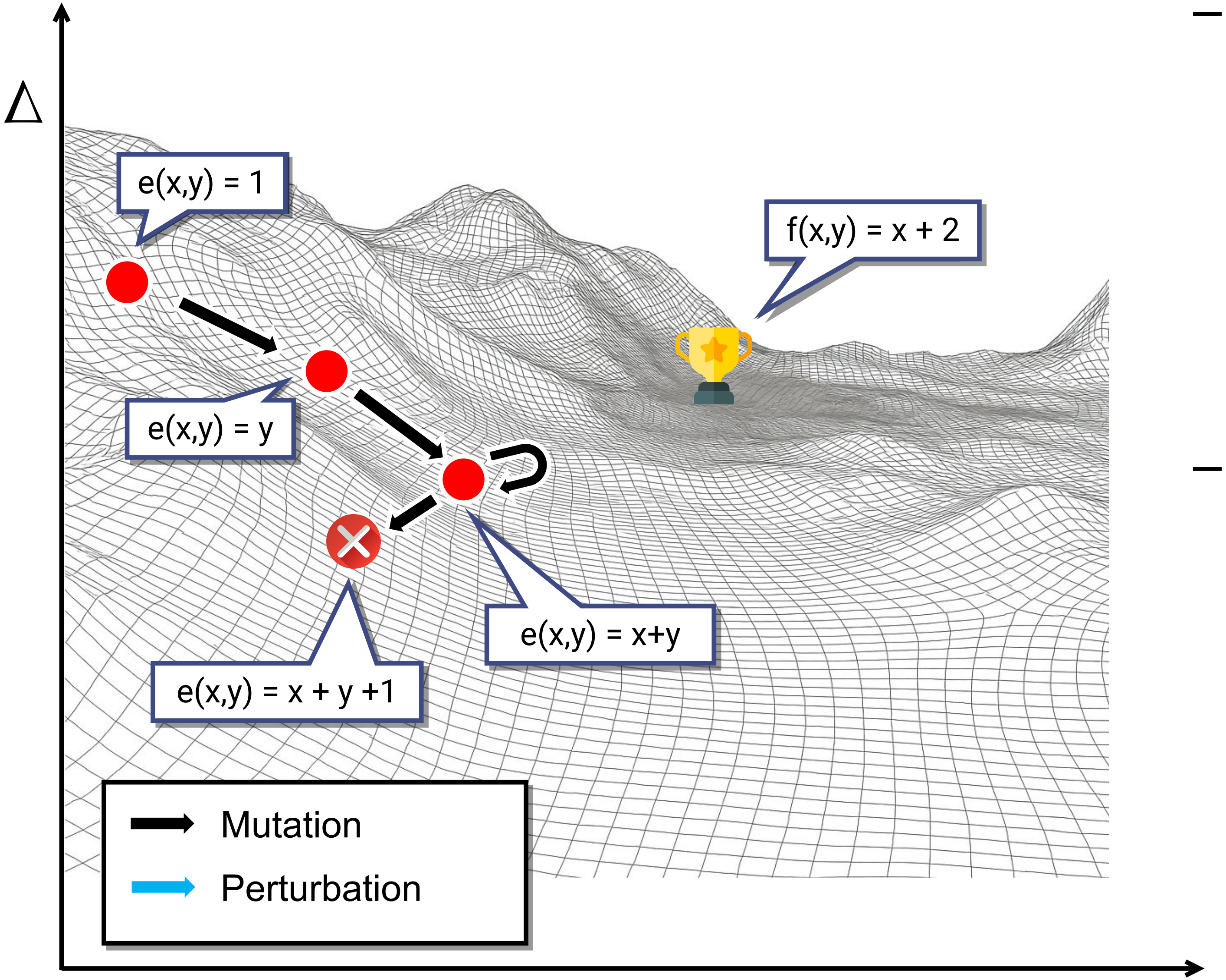
- **Guidance w.r.t., objective function**

$$\Delta(f, e) = \sum_{i \in Samples} |\overbrace{f(i)}^{\text{Target expr. output}} - \underbrace{e(i)}_{\text{Candidate expr. output}}|$$

- **Main search steps:**

1. **Mutations:** keep candidate only if Δ decreases
2. **Perturbations:** keep candidate even if Δ increases
3. **Ending condition:** if Δ = 0 we found the solution

Search-based Synthesis



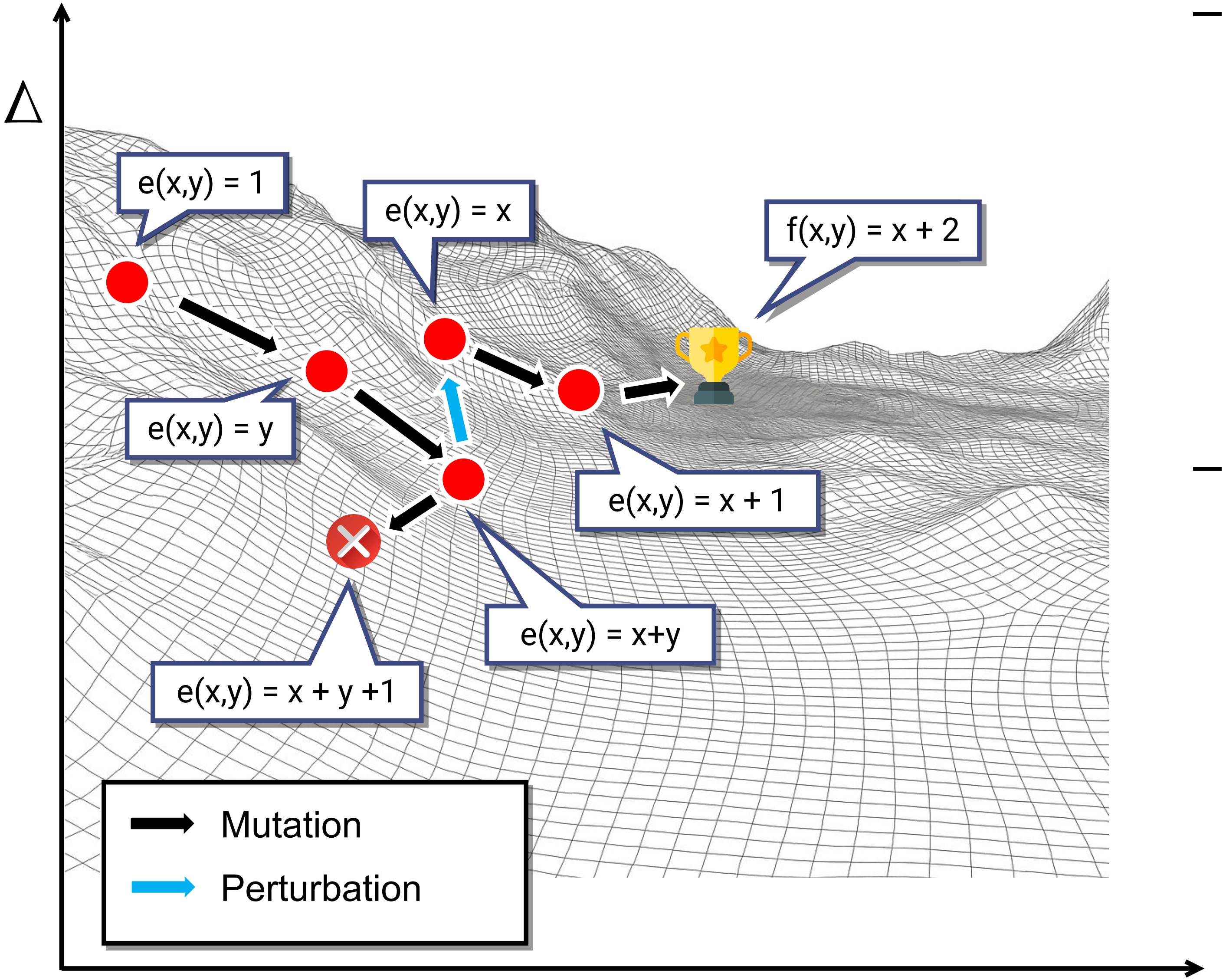
- **Guidance w.r.t., objective function**

$$\Delta(f, e) = \sum_{i \in Samples} |\overbrace{f(i)}^{\text{Target expr. output}} - \underbrace{e(i)}_{\text{Candidate expr. output}}|$$

- **Main search steps:**

1. **Mutations:** keep candidate only if Δ decreases
2. **Perturbations:** keep candidate even if Δ increases
3. **Ending condition:** if Δ = 0 we found the solution

Search-based Synthesis



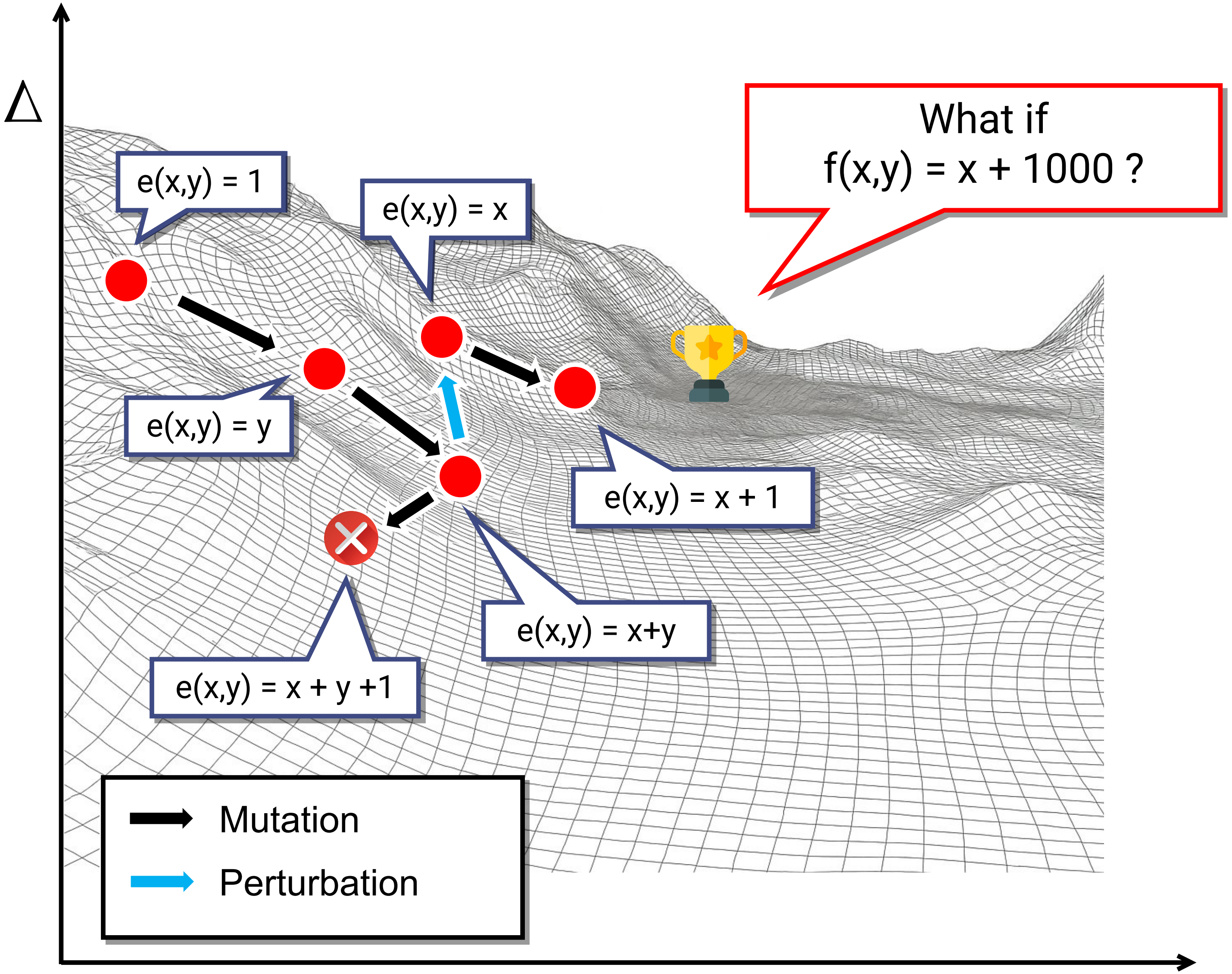
- **Guidance w.r.t., objective function**

$$\Delta(f,e) = \sum_{i \in Samples} \left|\overbrace{f(i)}^{\text{Target expr. output}} - \underbrace{e(i)}_{\text{Candidate expr. output}}\right|$$

- **Main search steps:**

1. **Mutations:** keep candidate only if Δ decreases
2. **Perturbations:** keep candidate even if Δ increases
3. **Ending condition:** if Δ = 0 we found the solution

Search-based Synthesis



What if
f(x,y) = x + 1000 ?

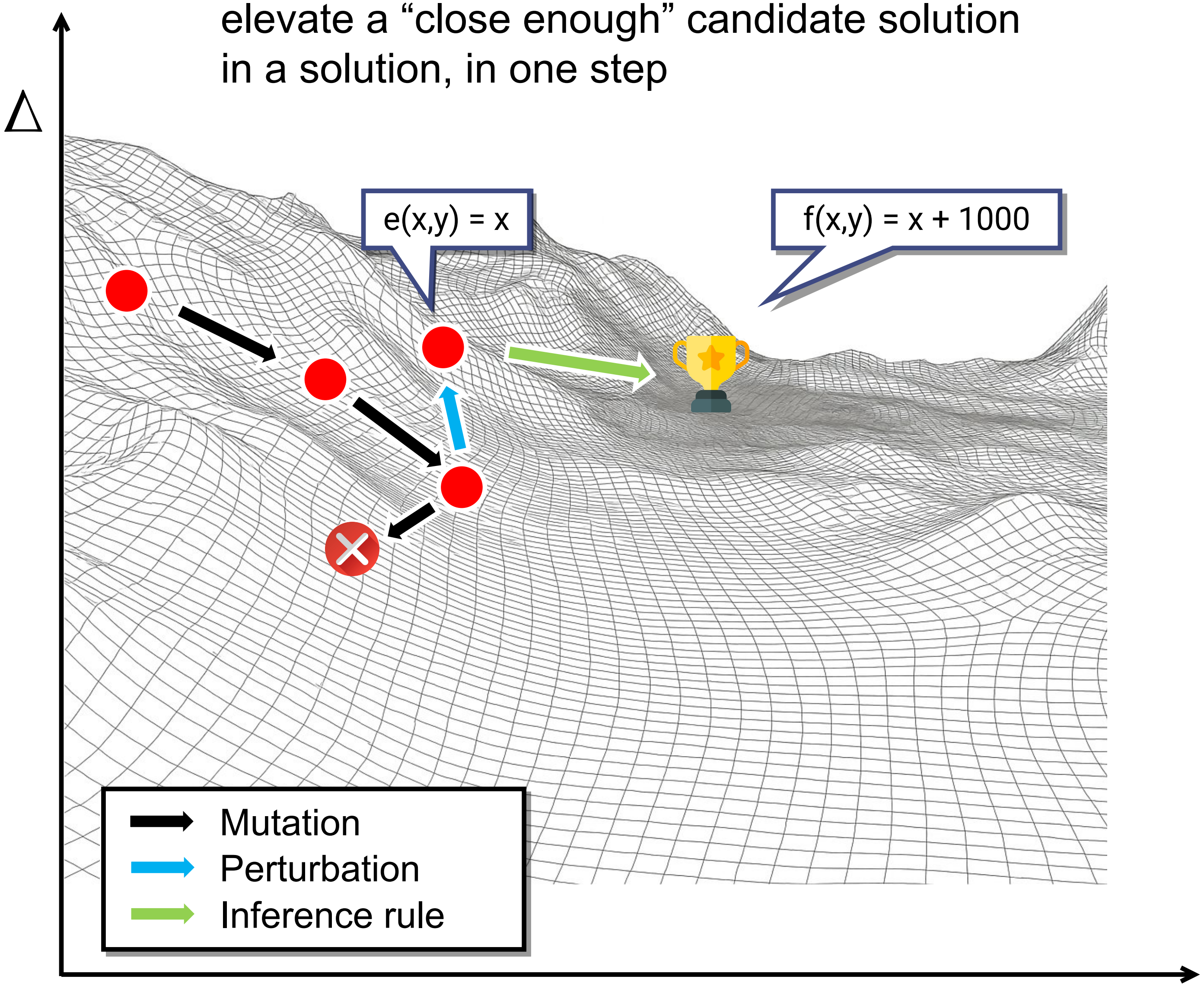
Problem:

- Very unlikely to generate $x + \underbrace{1 + \dots + 1}_{\text{times 1000}}$
- Search is too heuristic

Search Modulo Inference Rules (SMIR)

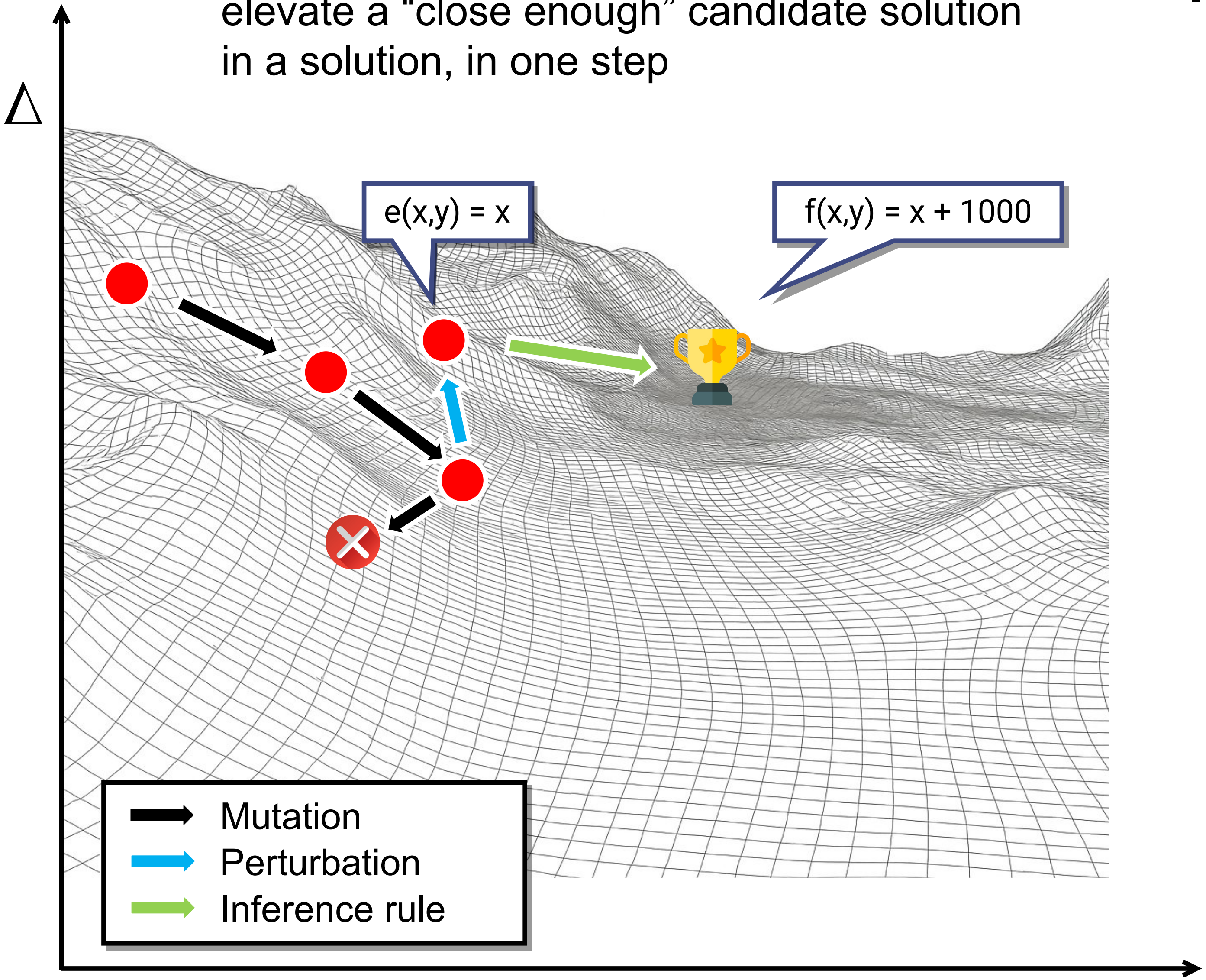
➡ Inference rule

elevate a “close enough” candidate solution in a solution, in one step



Search Modulo Inference Rules (SMIR)

➡ Inference rule
elevate a “close enough” candidate solution
in a solution, in one step

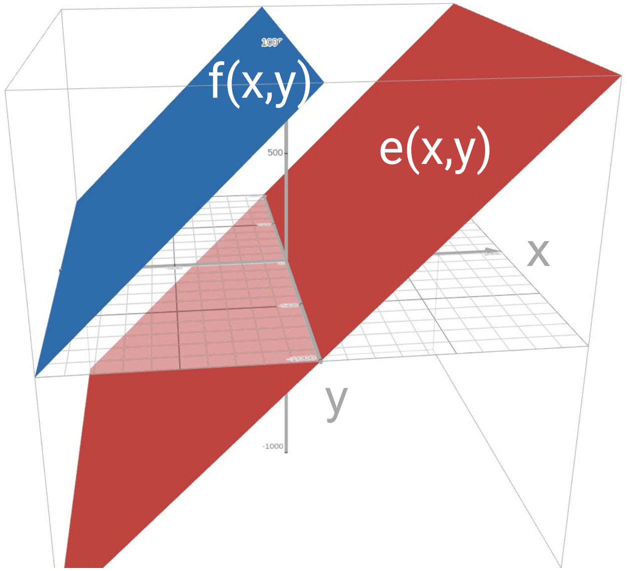


e(x,y) = x

f(x,y) = x + 1000

➡ Mutation
➡ Perturbation
➡ Inference rule

Inference rule for +



The variance of f(x,y) - e(x,y) equals 0

⇨ The solution equals e(x,y) up to an offset

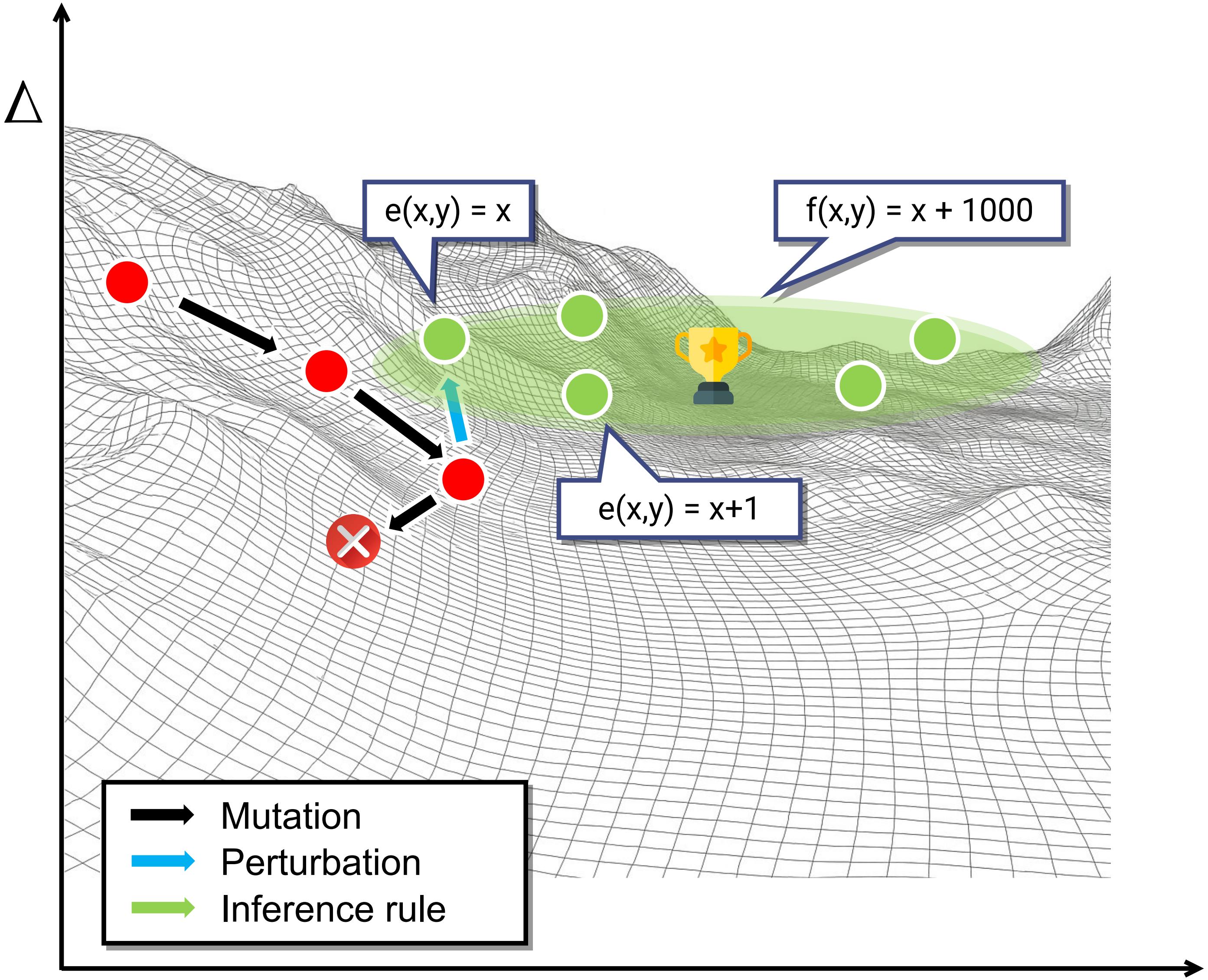
⇨ **Solution** = e(x,y) + f(0, 0) - e(0, 0)

$$\boxed{= \; e(x, y) + 1000}$$

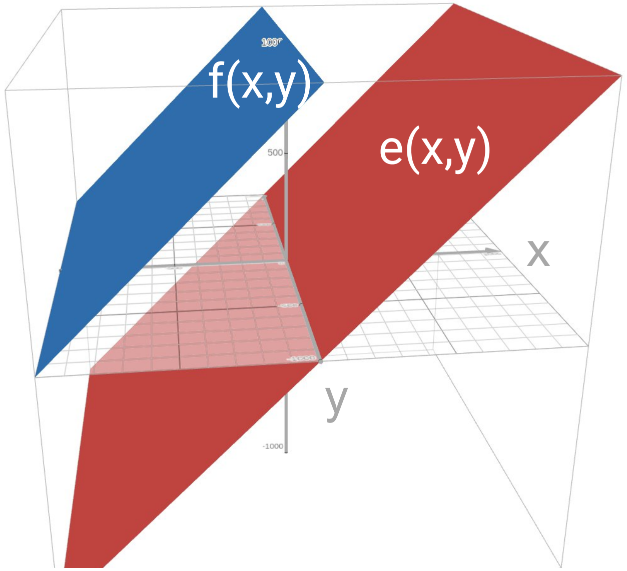
Search Modulo Inference Rules (SMIR)

Search: find the exact right expression 🏆

SMIR: find any close-enough expression ●



Inference rule for +



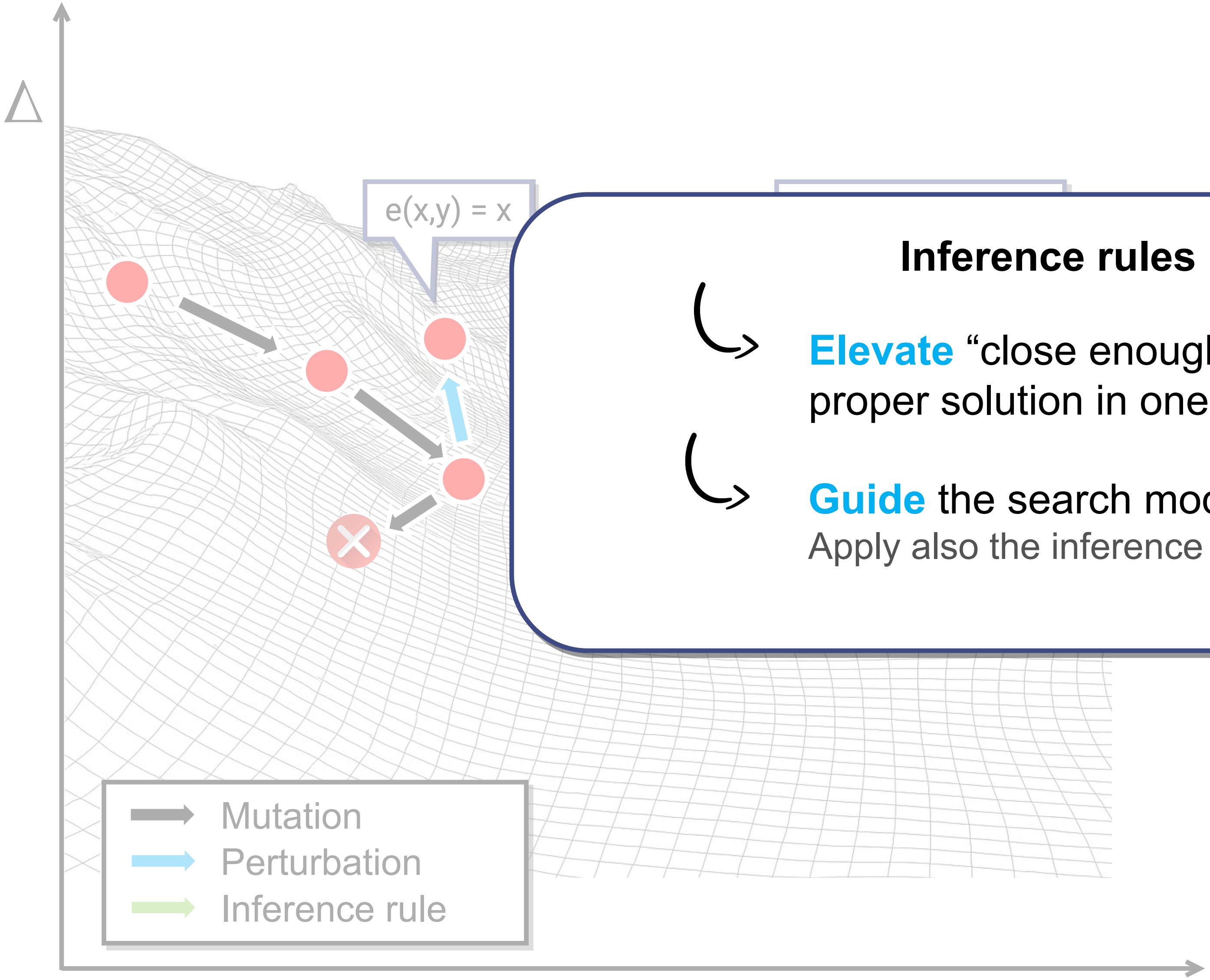
The variance of f(x,y) - e(x,y) equals 0

⇨ The solution equals e(x,y) up to an offset

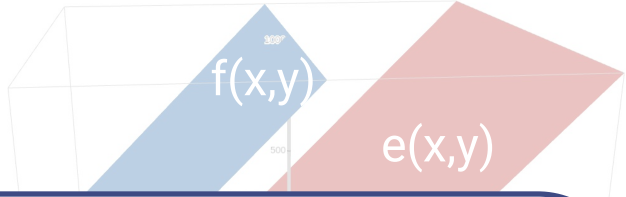
⇨ **Solution** = e(x,y) + f(0, 0) - e(0, 0)

$$\boxed{= \ e(x, y) + 1000}$$

Search Modulo Inference Rules (SMIR)



Inference rule for +



Inference rules usages

Elevate “close enough” candidate to a proper solution in one step

Guide the search modulo the rule

Apply also the inference rule to get better candidates

y) equals 0

⇨ The solution equals e(x,y) up to an offset

⇨ **Solution** = e(x,y) + f(0, 0) - e(0, 0)

$\boxed{= \ e(x, y) + 1000}$

Our Current Inference Rules

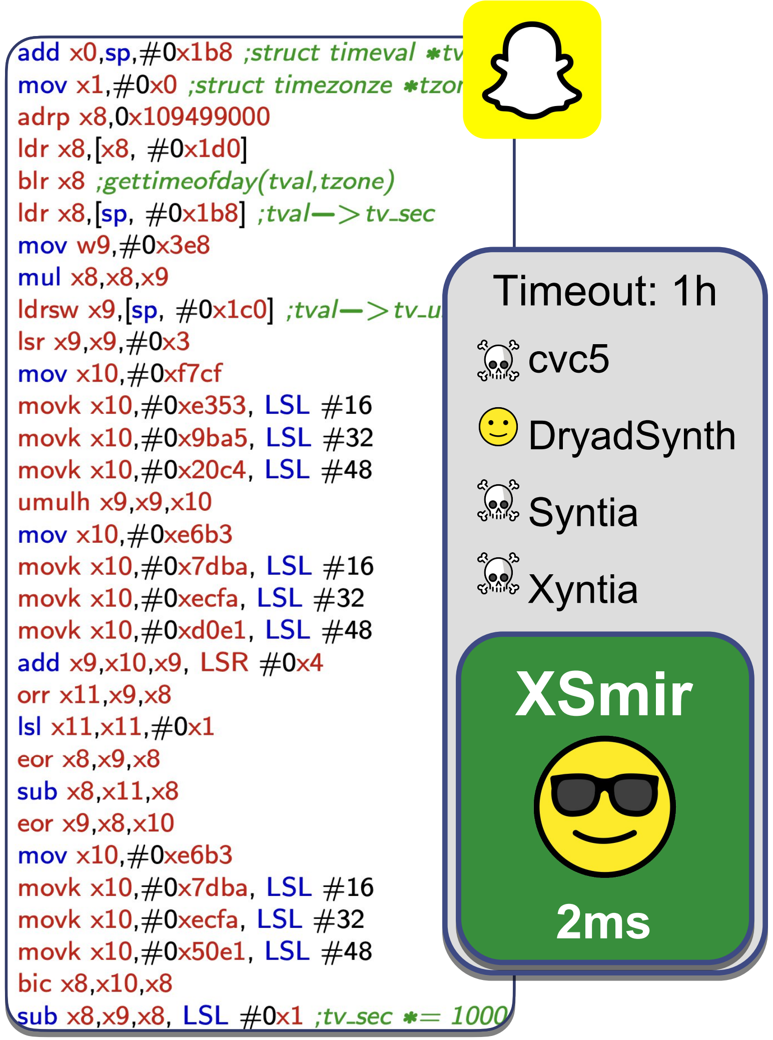
1	**Addition**	$e + c$
2	**Multiplication**	$e \times c$
3	**XOR mask**	$e \oplus c$
4	**AND/OR mask**	$(e \wedge c_1) \vee c_2$
5	**Rotation**	$rotate(e,c)$
6	**Log. left shift**	$e \ll c$
7	**Log. right shift**	$e \gg c$
8	**Affine**	$c_1 \cdot e + c_2$
9	**Polynomial**	$\sum_{i=0}^{k} c_i \cdot e^i$

In the paper:

- Recipes for creating rules
  > Inversion vs Enumeration
- Explain how to handle multiple rules

Back to our Examples

Real example from Snapchat iOS app



```
add x0,sp,#0x1b8 ;struct timeval *tv
mov x1,#0x0 ;struct timezonze *tzor
adrp x8,0x109499000
ldr x8,[x8, #0x1d0]
blr x8 ;gettimeofday(tval,tzone)
ldr x8,[sp, #0x1b8] ;tval->tv_sec
mov w9,#0x3e8
mul x8,x8,x9
ldrsw x9,[sp, #0x1c0] ;tval->tv_u
lsr x9,x9,#0x3
mov x10,#0xf7cf
movk x10,#0xe353, LSL #16
movk x10,#0x9ba5, LSL #32
movk x10,#0x20c4, LSL #48
umulh x9,x9,x10
mov x10,#0xe6b3
movk x10,#0x7dba, LSL #16
movk x10,#0xecfa, LSL #32
movk x10,#0xd0e1, LSL #48
add x9,x10,x9, LSR #0x4
orr x11,x9,x8
lsl x11,x11,#0x1
eor x8,x9,x8
sub x8,x11,x8
eor x9,x8,x10
mov x10,#0xe6b3
movk x10,#0x7dba, LSL #16
movk x10,#0xecfa, LSL #32
movk x10,#0x50e1, LSL #48
bic x8,x10,x8
sub x8,x9,x8, LSL #0x1 ;tv_sec *= 1000
```

Timeout: 1h

☠ cvc5

☺ DryadSynth

☠ Syntia

☠ Xyntia

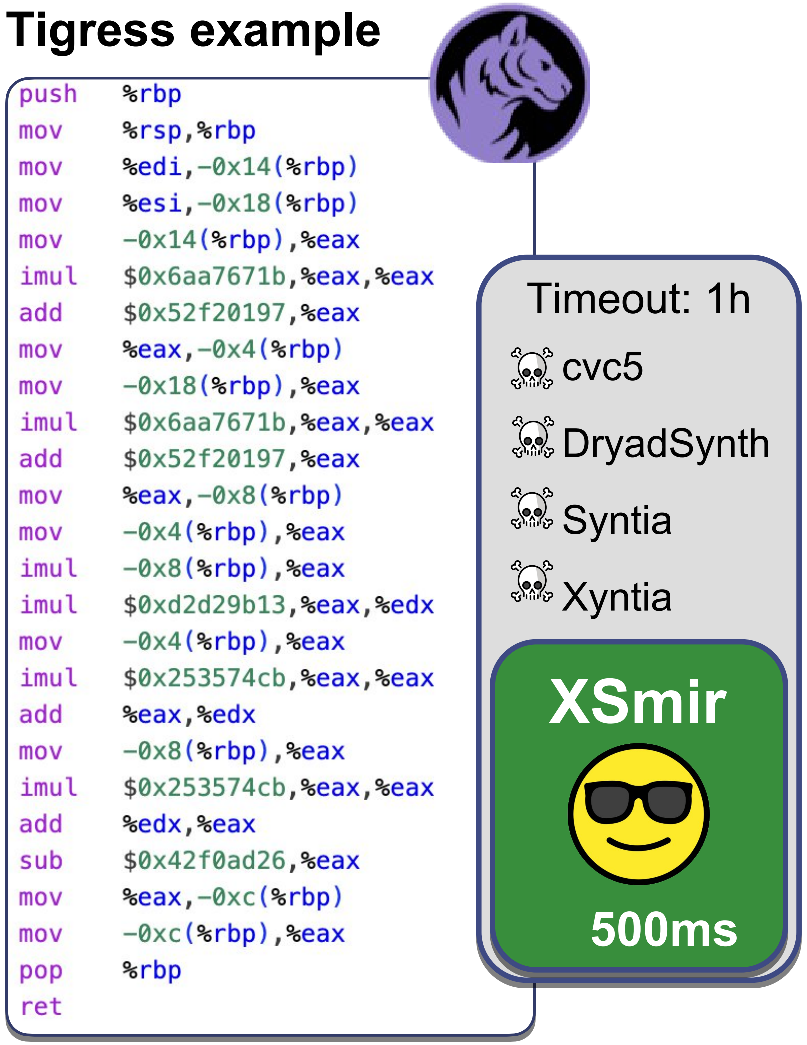
XSmir

😎

2ms

Expression: y = x * 1000

Tigress example



```
push    %rbp
mov     %rsp,%rbp
mov     %edi,-0x14(%rbp)
mov     %esi,-0x18(%rbp)
mov     -0x14(%rbp),%eax
imul    $0x6aa7671b,%eax,%eax
add     $0x52f20197,%eax
mov     %eax,-0x4(%rbp)
mov     -0x18(%rbp),%eax
imul    $0x6aa7671b,%eax,%eax
add     $0x52f20197,%eax
mov     %eax,-0x8(%rbp)
mov     -0x4(%rbp),%eax
imul    -0x8(%rbp),%eax
imul    $0xd2d29b13,%eax,%edx
mov     -0x4(%rbp),%eax
imul    $0x253574cb,%eax,%eax
add     %eax,%edx
mov     -0x8(%rbp),%eax
imul    $0x253574cb,%eax,%eax
add     %edx,%eax
sub     $0x42f0ad26,%eax
mov     %eax,-0xc(%rbp)
mov     -0xc(%rbp),%eax
pop     %rbp
ret
```

Timeout: 1h

☠ cvc5

☠ DryadSynth

☠ Syntia

☠ Xyntia

XSmir

😎

500ms

Expression: z = x * y * 0x6aa7671b + 0x52f20197

Evaluation: Research Questions

xyntia Public

Implementation of Xyntia/Smir (XSmir) as an extension of Xyntia

First evaluation of black-box deobfuscation at scale a full binaries

RQ1: Comparison with SotA on real world obfuscated binaries

RQ2: XSmir compression vs white-box deobfuscators on MBA

RQ3: Can XSmir recover diversified semantic expressions from in-the-wild binaries?

RQ4: How do internal parameters impact synthesis?

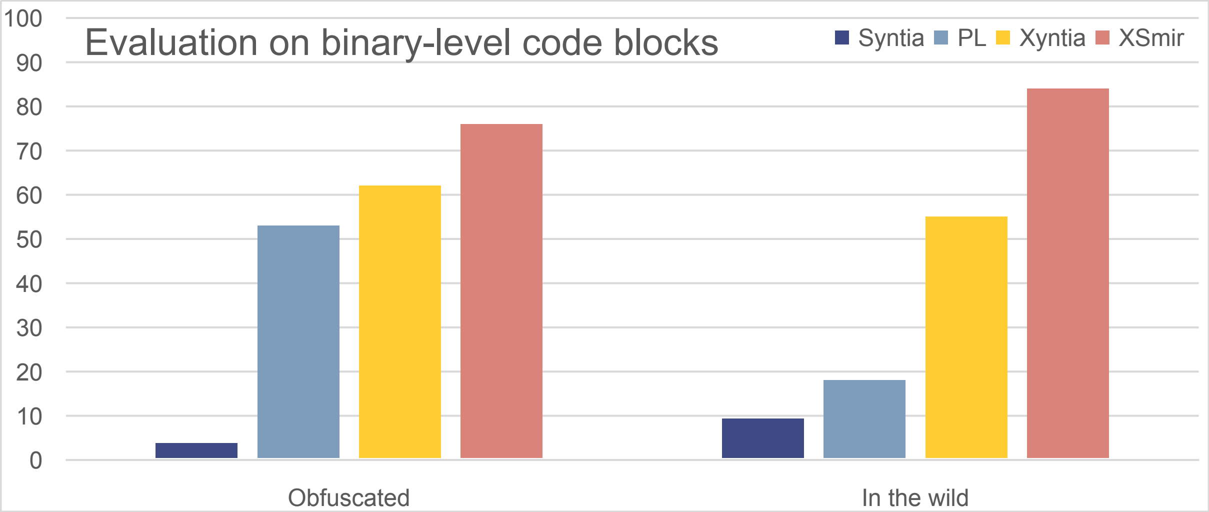
Ablation study

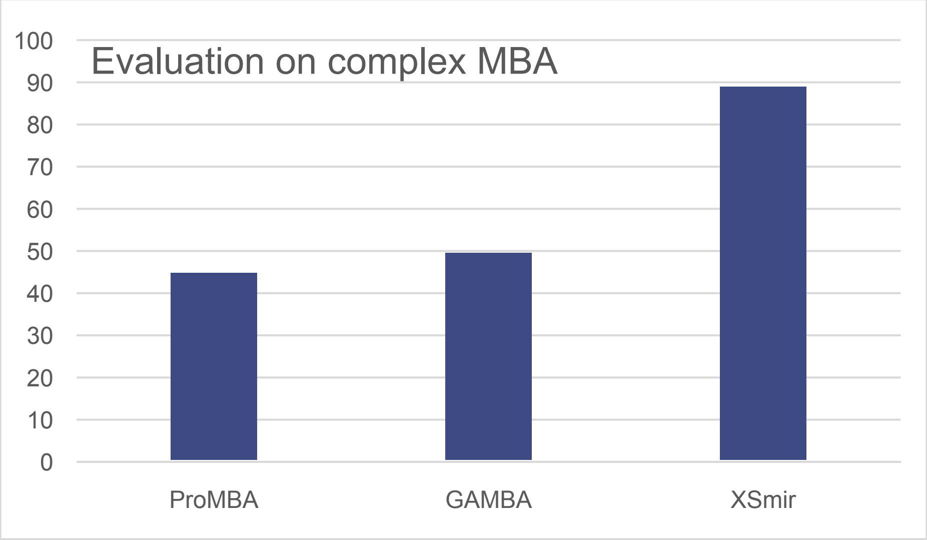
Impact of the objective function

Inference rules order

Impact of the rules combinator

Experimental Results Summary





Obfuscated binaries : **76%** vs. **63%** vs **53%** vs **4%**

In-the-wild binaries: **84%** vs **55%** vs **18%** vs **9%**

Complex MBA: **90%** vs. **50%**

⊕ XSmir is black-box and general purpose

Also: Divides by 2 the number of false positives

Conclusion

– Extension of Black-box deobfuscation

↳ SMIR: inference rules to guide the search & elevate expressions to the solution

↳ Inference rules for standard Bitvector operators, affine and polynomial relations

Black-box:
From VM-handlers to arbitrary code blocks

– First evaluation of black-box deobfuscation at scale on full binaries

↳ Outperforms prior black-box deobfuscators [Xyntia, Syntia]

↳ Outperforms synthesizers from the PL community [CVC4/5, DryadSynth]

↳ Better simplify MBA expressions than white-box specialized tools [ProMBA, GAMBA]

xyntia Public

https://github.com/binsec/xyntia

zenodo

Artifacts Available V1.1

Artifacts Evaluated Functional V1.1

Results Reproduced V1.1