$> Black-box Deobfuscation: Reverse Engineering Binaries with your Eyes Closed (or almost)

Speaker

Grégoire MENGUY (CEA LIST, France)

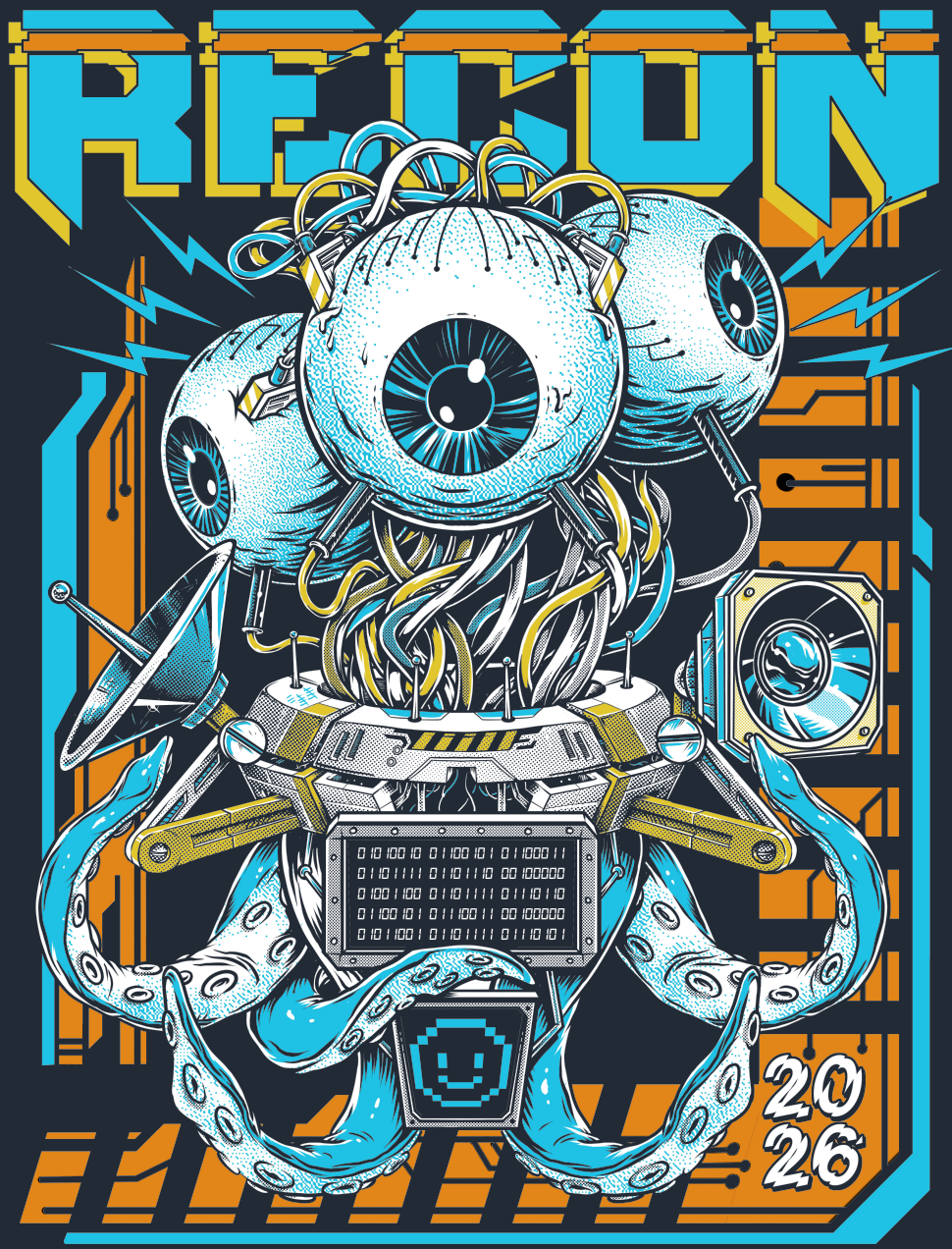
Joint work with:

Vidal Attias (CEA LIST, France)

Nicolas Bellec (CEA LIST, France)

Sébastien Bardin (CEA LIST, France)

Jean-Yves Marion (LORIA, France)



About Me

- Academic researcher
- **Topics:** Binary analysis for security, RE
- **Publications:** CCS, IJCAI etc
  > Security and AI conferences

Here

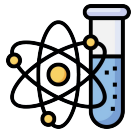
- **CEA:** French research institute
- **Where?** In the south of **Paris in France** 🇫🇷 🥐
- **PhD/ internships offers :** send me an message
  > https://gregoiremenguy.github.io/
  > https://binsec.github.io/

First time at Recon
> Its awesome

😍

The Talk in 30 secs

Introduction to black-box deobfuscation
> what and why ?



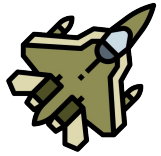
A glimpse in the black-box deobfuscation journey with Xyntia
> Guiding the synthesis
> The constant values case

Examples of deobfuscation
> MBA
> Virtualization

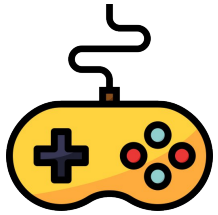
Obfuscation & Deobfuscation

Google play store :
- 25% of apps obfuscated
- 50% of most popular apps

Web apps:
- JS obfuscation
- Web assembly



Military:
- Hainan Island incident 2001



Video games:
- Anti-cheat

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

Obfuscation

☑ Intellectual property

☣ Thwart analysis and detection

```
igned int add_simple(unsigned int x ,
nsigned 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))))));
```

Deobfuscation

☣ Help malware analysis

☑ Check protections

Basic Examples of Obfuscation

Remark
I will mainly talk about data-flow obfuscation

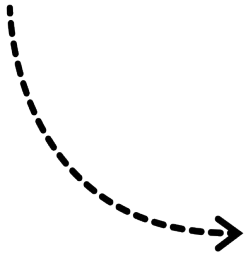
> Mixed Boolean Arithmetic (MBA)

```
int mba( int x, int y ) {
    // Computes x * y + x
    return (x&y)*(x|y)+(x&~y)*(~x&y)+x;
}
```

+

> Opaque predicate

```
int opaque( int x, int y ) {
    if ( x * y = x * y) x * y + x;
    else x * y - x;
}
```



```
int obfuscated( int x, int y ) {
    int a = (x&y)*(x|y)+(x&~y)*(~x&y);
    if ( a = x * y) return a + x;
    else a - x;
}
```

Combined obfuscations is hard to reverse

Basic Examples of Obfuscation

Remark
I will mainly talk about data-flow obfuscation

> Mixed Boolean Arithmetic (MBA)

```
int mba( int x, int y ) {
    // Computes x * y + y
    r
}
```

> Opaque predicate

```
int opaque( int x, int y ) {
    if ( a = x * y) return a + x;
    else a - x;
}
```

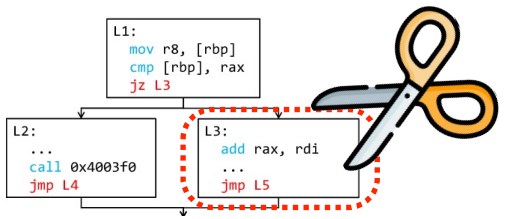
Deobfuscation immune to standard obfuscation?

> Results must not depend on the syntax of the code

Combined obfuscations is hard to reverse

Solution: Black-Box Deobfuscation

1 Defining a reverse window



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

Session 10A: Crypto, Symbols and Obfuscation CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

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

Richard Bonichon
richard.bonichon@nomadic-labs.com
Nomadic Labs
France

Cauim de Souza Lima
cauimsouza@gmail.com
Université Paris-Saclay, CEA, List
France

ABSTRACT

Code obfuscation aims at protecting Intellectual Property and other secrets embedded into software from being retrieved. Recent works leverage advances in artificial intelligence (AI) with the hope of getting blackbox deobfuscators completely immune to standard (whitebox) protection mechanisms. While promising, this new field of *AI-based*, and more specifically *search-based blackbox deobfuscation*, is still in its infancy. In this article we deepen the state of search-based blackbox deobfuscation in three key directions: *understand* the current state-of-the-art, *improve over* it and design dedicated *protection mechanisms*. In particular, we define a novel generic framework for search-based blackbox deobfuscation encompassing prior work and highlighting key components; we are the first to point out that the search space underlying code deobfuscation is too unstable for simulation-based methods (e.g., Monte Carlo Tree Search used in prior work) and advocate the use of robust methods such as S-metaheuristics; we propose the new optimized search-based blackbox deobfuscator Xyntia which significantly outperforms prior work in terms of success rate (especially with small time budget) while being completely immune to the most recent anti-analysis code obfuscation methods; and finally we propose two novel protections against search-based blackbox deobfuscation, allowing to counter Xyntia powerful attacks.

CCS CONCEPTS

• Security and privacy → Software reverse engineering; • Computing methodologies → Randomized search; Game tree search.

KEYWORDS

Binary-level code analysis; deobfuscation; artificial intelligence; S-metaheuristics

ACM Reference Format:
Grégoire Menguy, Sébastien Bardin, Richard Bonichon, and Cauim de Souza Lima. 2021. Search-Based Local Black-Box Deobfuscation: Understand, Improve and Mitigate. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3460120.3485250

1 INTRODUCTION

Context. Software contain valuable assets, such as secret algorithms, business logic or cryptographic keys, that attackers may try to retrieve. The so-called Man-At-The-End-Attacks scenario (MATE) considers the case where software users themselves are adversarial and try to extract such information from the code. *Code obfuscation* [12, 13] aims at protecting codes against such attacks, by transforming a sensitive program P into a functionally equivalent program P' that is more "difficult" (more expensive, for example, in money or time) to understand or modify. On the flip side, *code deobfuscation* aims to extract information from obfuscated codes.

Whitebox deobfuscation techniques, based on advanced symbolic program analysis, have proven extremely powerful against standard obfuscation schemes [3, 5, 10, 22, 28, 30, 36] – especially in local attack scenarios where the attacker analyses pre-identified parts of the code (e.g., trigger conditions). But they are inherently sensitive to the *syntactic complexity* of the code under analysis, leading to recent and effective countermeasures [12, 25, 26, 37].

Search-based blackbox deobfuscation. Despite being rarely complete or sound, *artificial intelligence* (AI) techniques are flexible and often provide good enough solutions to hard problems in reasonable time. They have been therefore recently applied to binary-level code deobfuscation. The pioneering work by Blazytko et al. [7] shows how *Monte Carlo Tree Search* (MCTS) [9] can be leveraged to solve local deobfuscation tasks by *learning* the semantics of pieces of protected codes in a *blackbox manner*, in principle *immune to the syntactic complexity* of these codes. Their method and prototype, Syntia, have been successfully used to reverse state-of-the-art protectors like VMProtect [34], Themida [27] and Tigress [11], drawing attention from the software security community [8].

Problem. While promising, search-based blackbox (code) deobfuscation techniques are still not well understood. Several key questions of practical relevance (e.g., deobfuscation correctness and quality, sensitivity to time budget) are not addressed in Blazytko et

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3485250

What is Program Synthesis ?

Here no LLM nor neural nets
> No need for training

Syntactic specification

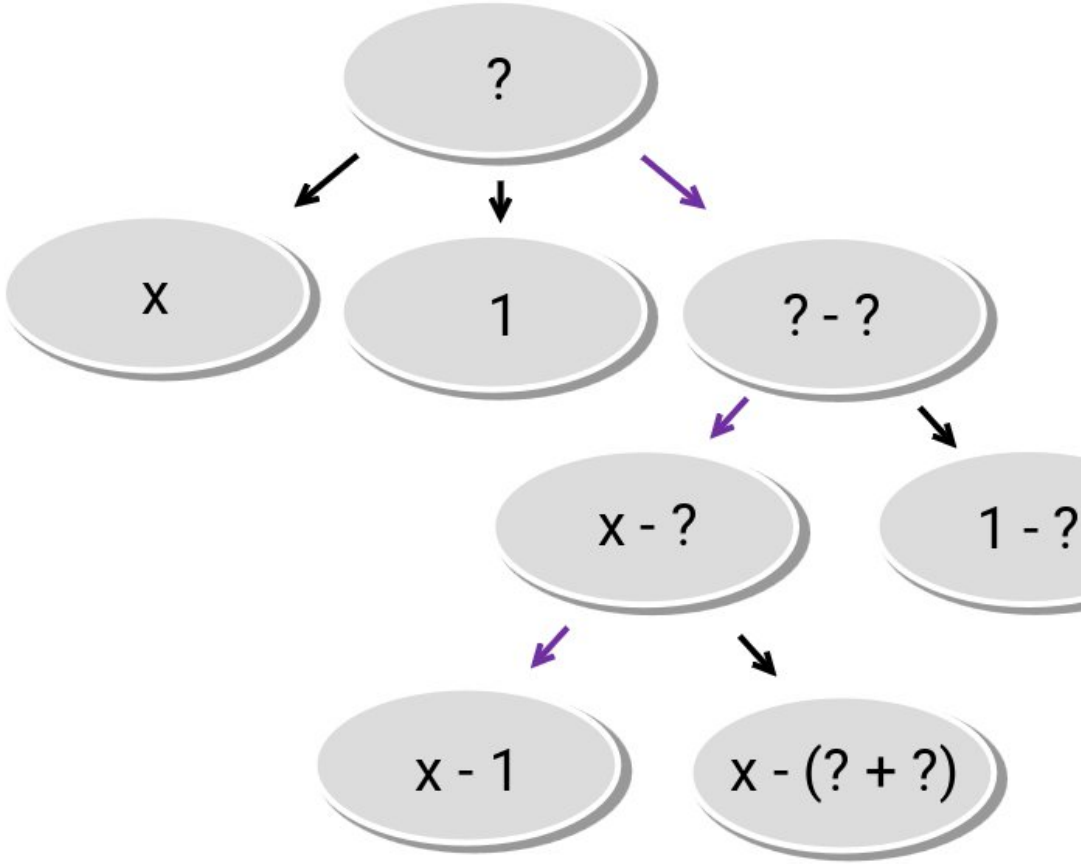
E := E + E | E - E | ...
E := V | C
V := x
C := 1

+

Semantic specification

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

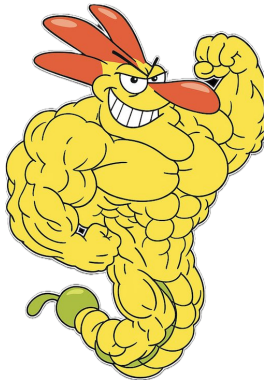
SyGuS
Syntax-Guided Synthesis



Immune to Syntactic Complexity

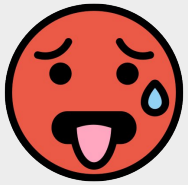
```
int add(int x , int y )
{
  return ((x ^ y) +
   ((x & y) << 1));
}
```

```
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

Lets summarize

Black-box deobfuscation

Goal: simplify **local** and **highly obfuscated** code snippets

Does not use the syntax
> only input-output observations
> Immune to obfuscation

Core technology : Program synthesis

Help for reverse engineers
> You tell where is the obfuscated part
> It deobfuscates it for you

A Glimpse in Black-box Deobfuscation with Xyntia

binsec / xyntia Public

77 stars

What is Xyntia ?

> Black-box deobfuscator

> Binary code analysis

> Sampler + Synthesizer

> Open source (LGPL 2.1)

https://github.com/binsec/xyntia

Key Challenge

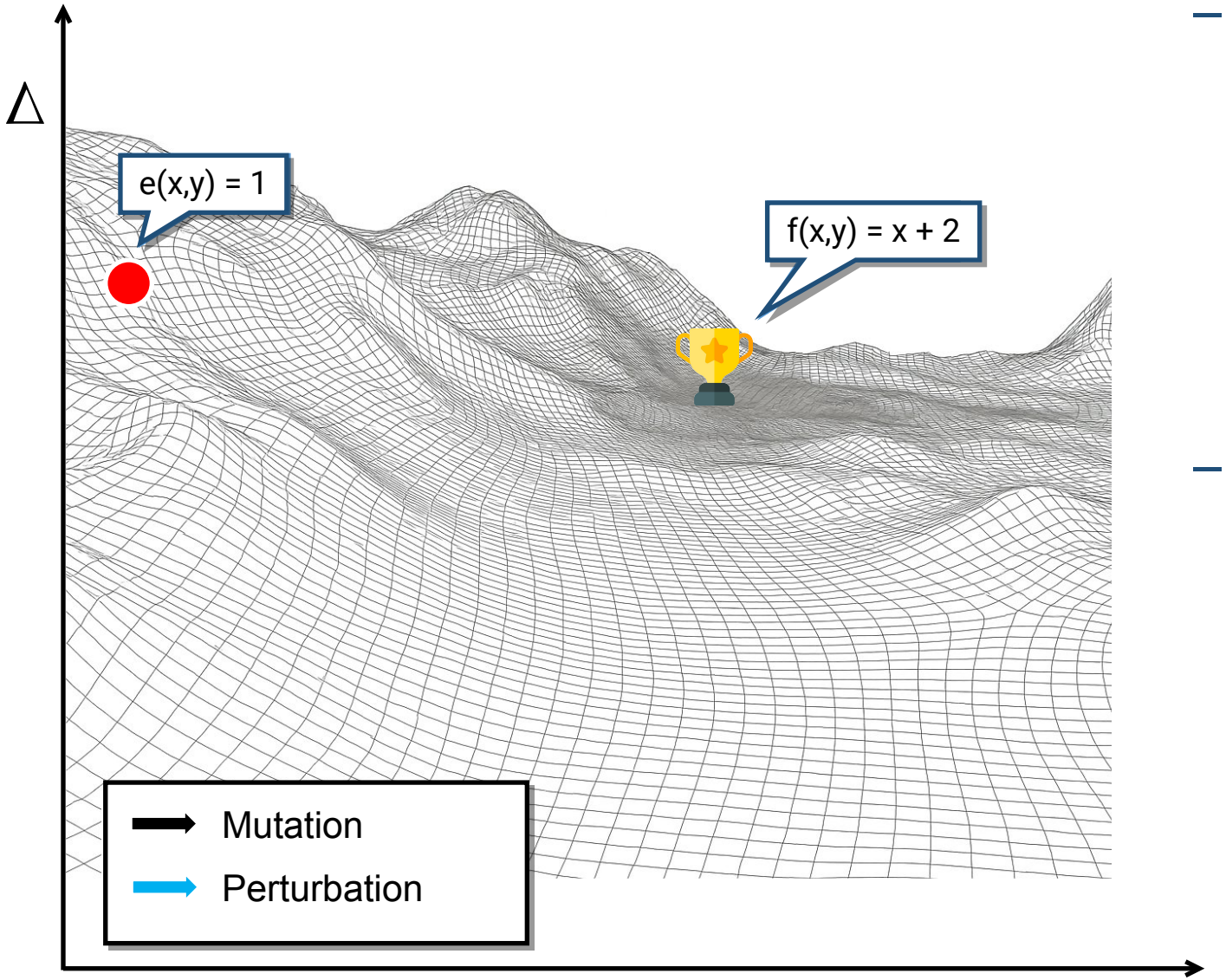
Get powerful enough synthesis to handle real obfuscated code

Problem: SOTA is not enough

SYNTIA CVC4 CVC5

Goal of Xyntia: Provide Program Synthesis on Steroids
> For efficient deobfuscation

Xyntia: Synthesis as an Optimization Problem



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

Target expr. output

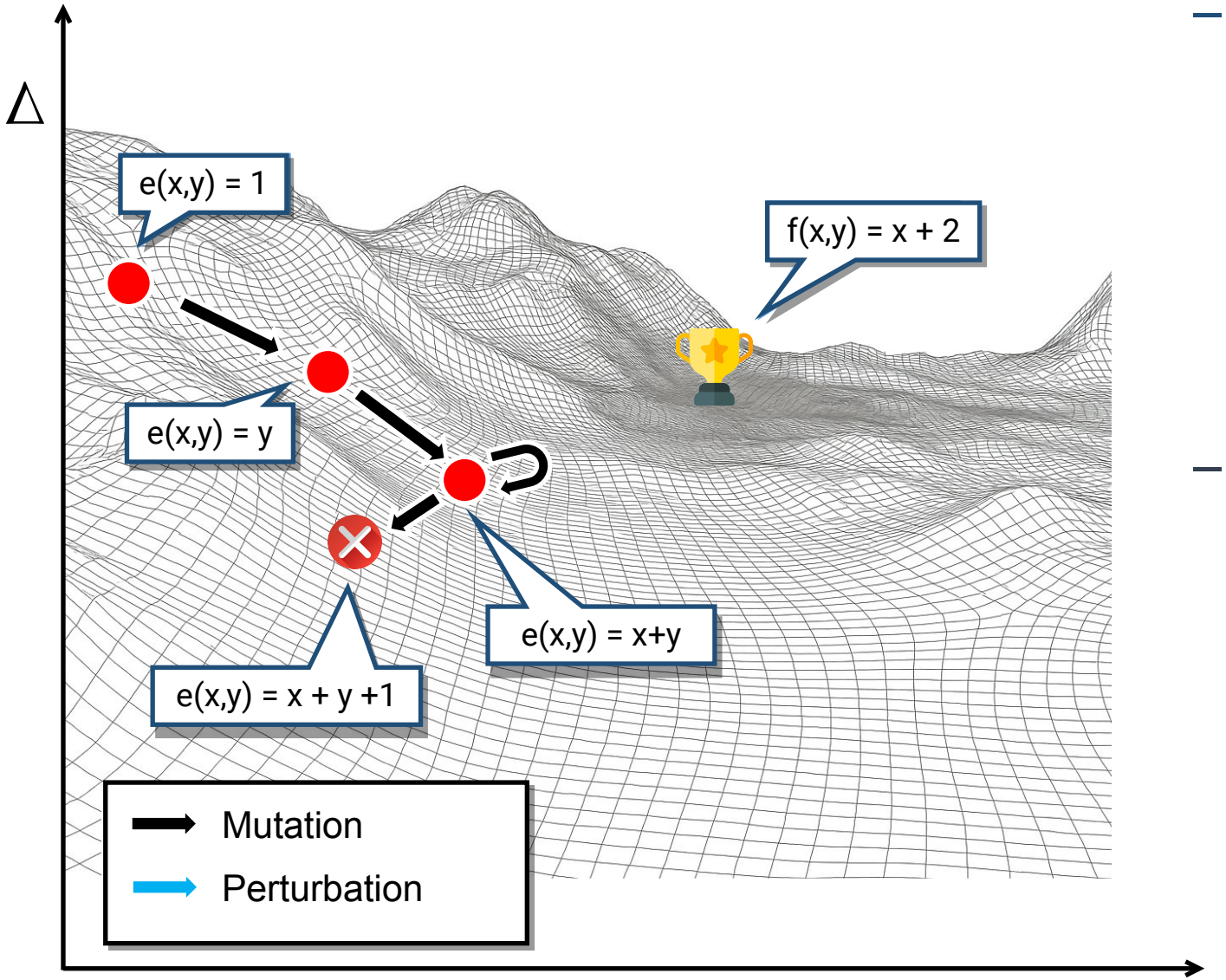
$$\Delta(f, e) = \sum_{i \in Samples} |f(i) - e(i)|$$

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

Xyntia: Synthesis as an Optimization Problem



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

Target expr. output

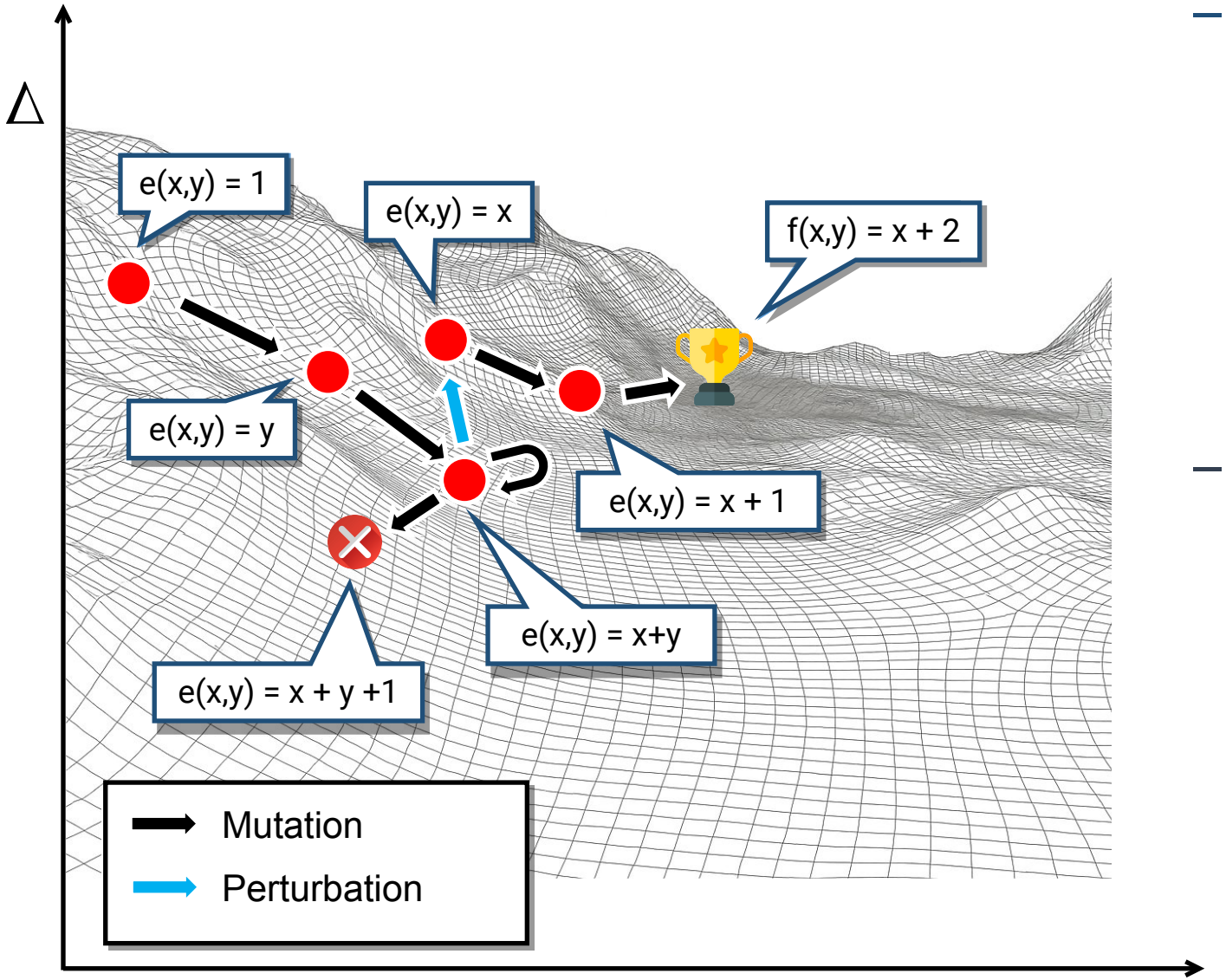
$$\Delta(f, e) = \sum_{i \in Samples} |\overbrace{f(i)}^{} - \underbrace{e(i)}_{}|$$

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

Xyntia: Synthesis as an Optimization Problem



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

Target expr. output

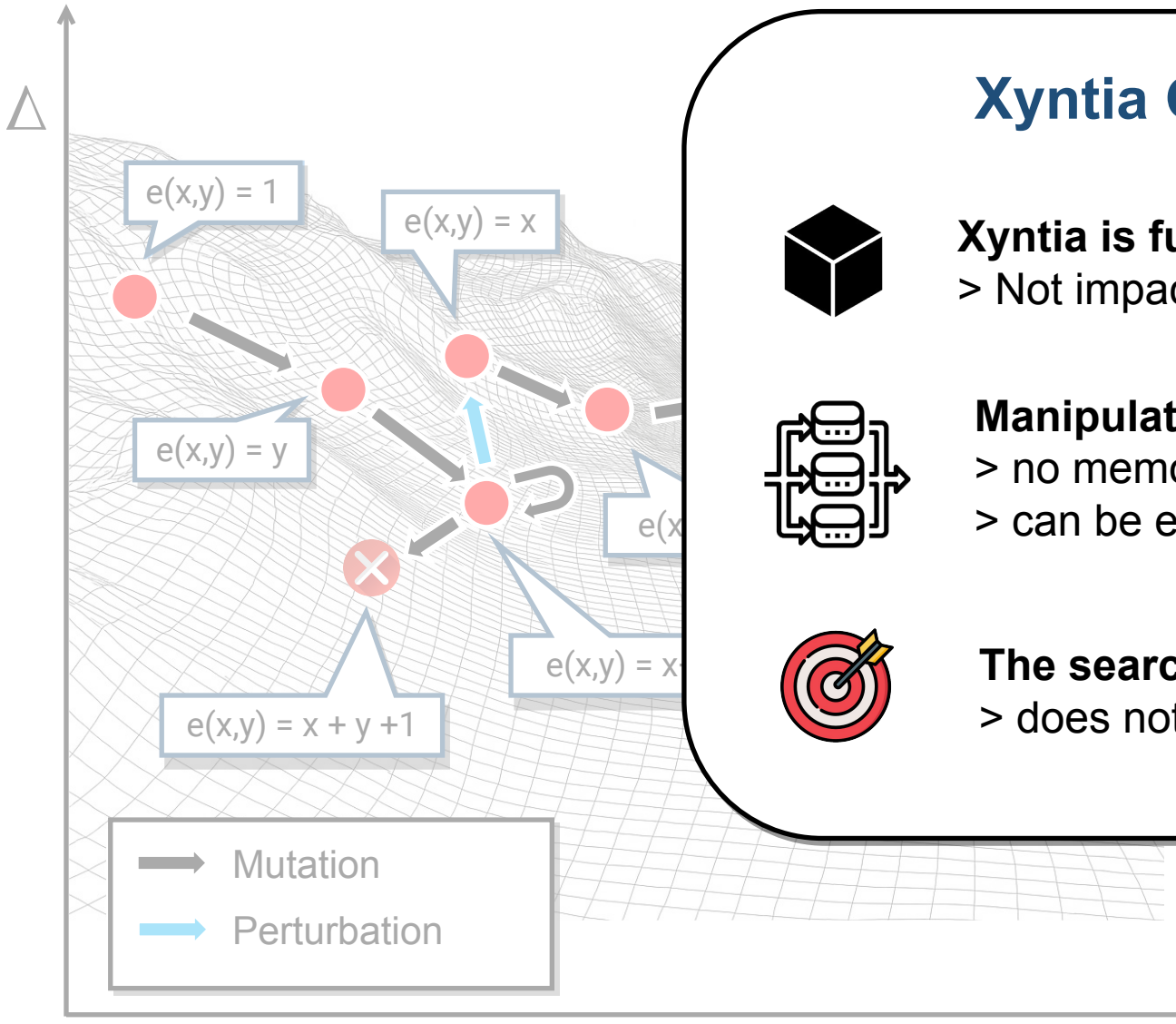
$$\Delta(f, e) = \sum_{i \in Samples} |\overbrace{f(i)}^{} - \underbrace{e(i)}_{}|$$

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

Xyntia: Synthesis as an Optimization Problem



Xyntia Good properties

Xyntia is full black-box
> Not impacted by standard obfuscation

Manipulates only one expression
> no memory exhaustion
> can be easily run in parallel

The search is guided
> does not enumerate all the search space

Back to our Examples

Mixed Boolean Arith. (MBA)

```
int mba( int x, int y ) {
    return (x&y)*(x|y)+(x&~y)*(~x&y)+x;
}
```

x * y + x

Opaque Predicate

```
int opaque( int x, int y ) {
    if ( x * y = x * y) x * y + x;
    else x * y - x;
}
```

x * y + x

MBA + Opaque Predicate

```
int obfuscated( int x, int y ) {
    int a = (x&y)*(x|y)+(x&~y)*(~x&y);
    if ( a = x * y) return a + x;
    else a - x;
}
```

x * y + x

Have we solved deobfuscation?

Still some cool research to do ❤

Clearly, black-box deobfuscation is **powerful**

But, it also comes with limitations

- Black-box is impacted by **semantical complexity**
  > Some expressions are too hard to synthesize
- In such a case, synthesis never ends ⏱

Example of problem

Real example from Snapchat iOS app

```
add x0,sp,#0x1b8 ;struct timeval *tva
mov x1,#0x0 ;struct timezonze *tzone
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

- cvc4/5
- 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,%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

- cvc4/5
- DryadSynth
- Syntia
- Xyntia

Expression:

$z = x * y * 0x6aa7671b + 0x52f20197$

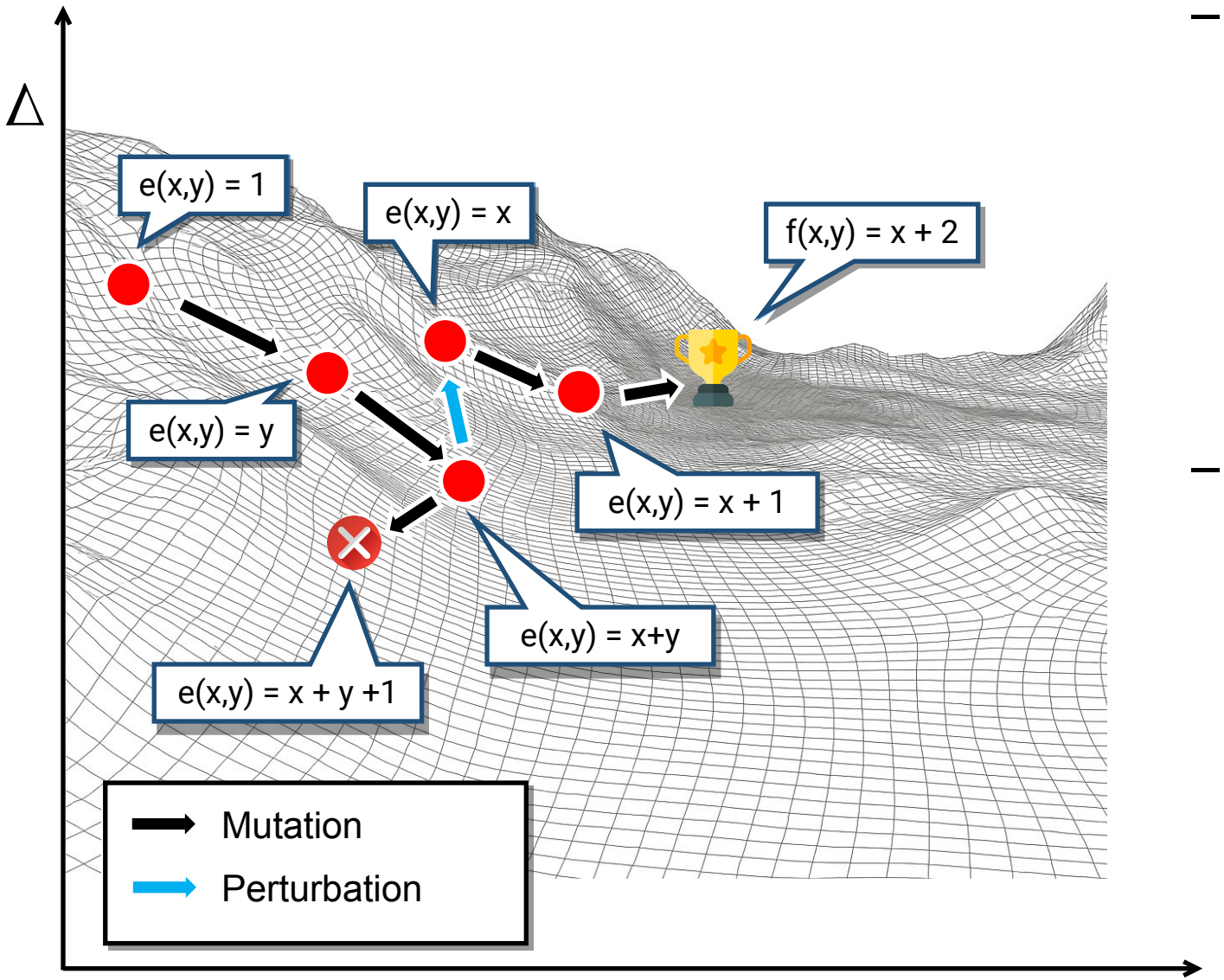
Common limitations in SyGUS

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

But there are limitations

- Constant values
- Large expressions

Why ?



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

Target expr. output

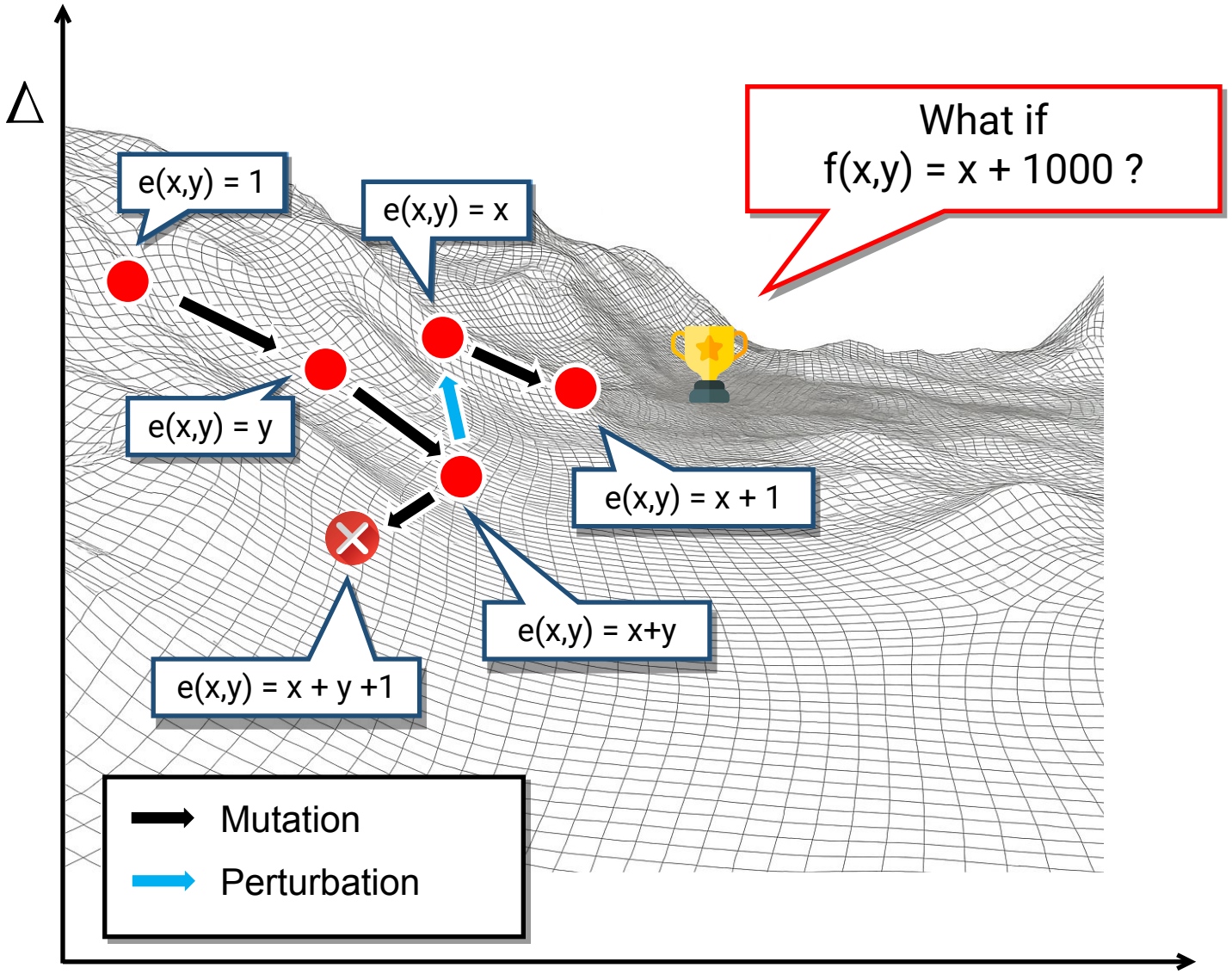
$$\Delta(f, e) = \sum_{i \in Samples} |\overbrace{f(i)}^{} - \underbrace{e(i)}_{}|$$

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

Why ?



Problem:

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

News: Search Modulo Inference Rules

1. **Search Modulo Inference Rules (SMIR)**

   Combine **search-based synthesis** with **fast symbolic reasoning**
   - Automatically *elevate* candidate solutions
   - Guide the search

2. **Implementation of SMIR on top of Xyntia**

   https://github.com/binsec/xyntia

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

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

Vidal Attias
Université Paris-Saclay, CEA, List
Palaiseau, France
vidal.attias@cea.fr

Nicolas Bellec
Université Paris-Saclay, CEA, List
Palaiseau, France
nicolas.bellec@cea.fr

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

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

Jean-Yves Marion
Université de Lorraine, CNRS, LORIA
Nancy, France
jean-yves.marion@loria.fr

Abstract

Code obfuscation aims to protect programs from reverse engineering, with applications ranging from intellectual property protection to malware hardening. Recent works on black-box analyses propose to leverage program synthesis in order to infer the semantics of highly obfuscated code blocks. Being fully *black-box*, these approaches are immune to *syntactic* complexity and can thus bypass standard obfuscation mechanisms. Yet, they are restricted by their synthesis capabilities and can only be applied to *semantically* simple code blocks. It explains why they have mainly been used on virtual machine handlers, where behaviors are usually simple enough. Applying black-box deobfuscation at scale beyond virtualization is still an open problem, notably because black-box methods cannot synthesize complex behaviors involving, for example, arbitrary constant values or affine or polynomial relations over mixed-boolean-arithmetic expressions. In this article, we show how to combine *search-based program synthesis* with *local inference rules*, resulting in a new method named *Search Modulo Inference Rules* (SMIR) that boosts search-based program synthesis while keeping its generality and flexibility. We instantiate SMIR with inference rules for hard synthesis problems like arbitrary constant values and affine or polynomial relations over mixed boolean expressions, yielding the new black-box deobfuscation tool: XSMIR. Experiments demonstrate that XSMIR significantly outperforms prior black-box deobfuscators, synthesizing overall 76% and 84% of the expressions from our real-world obfuscated and non-obfuscated benchmarks where prior works recover 63% and 55%, together with 2 to 3 times less false positive and slightly improved compression rate.

CCS Concepts

• **Security and privacy** → **Software reverse engineering**; • **Computing methodologies** → **Randomized search**.

Keywords

Deobfuscation, reverse engineering, program synthesis

CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765134

ACM Reference Format:
Vidal Attias, Nicolas Bellec, Grégoire Menguy, Sébastien Bardin, and Jean-Yves Marion. 2025. Augmenting Search-based Program Synthesis with Local Inference Rules to Improve Black-box Deobfuscation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765134

1 Introduction

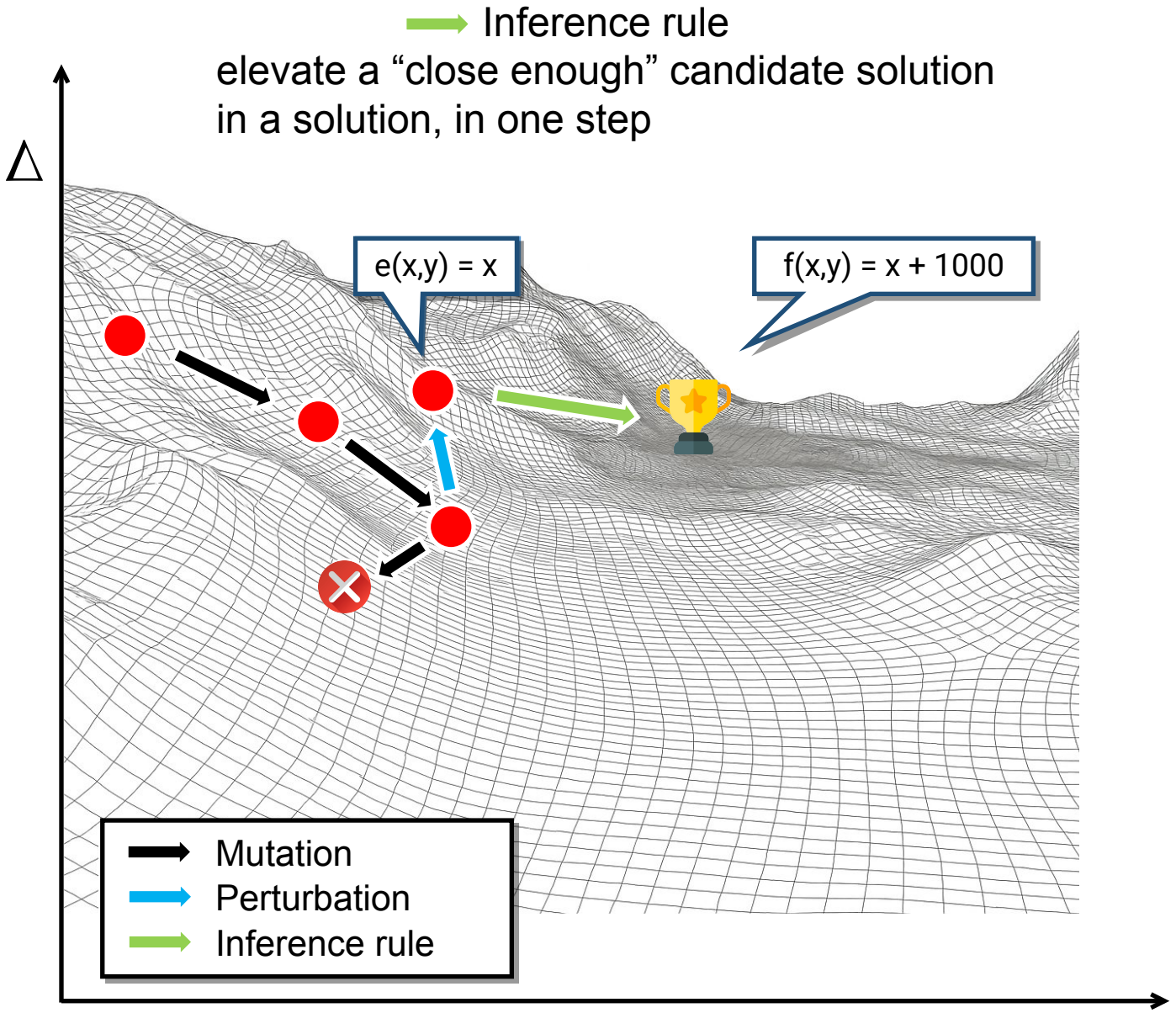
Obfuscation [18, 19] aims to protect software from reverse engineering. It translates a program P into a functionally equivalent program P_o, harder to analyze. As cryptography has not yet provided a bullet-proof solution for this problem, the obfuscation community mainly focuses on program-analysis-based techniques [19], leading to a cat-and-mouse game between defenders and attackers.

While obfuscation is used to protect *Intellectual Property* and other valuable software assets, it is also used to protect malware. Thus, automated *deobfuscation* methods [12, 21, 39, 50, 51] have been proposed to cope with the quick advances in obfuscation. Given an obfuscated program P_o, the goal here is to simplify it into a simpler yet functionally equivalent program P^* – ideally P^* should be as simple as the original unprotected code P.

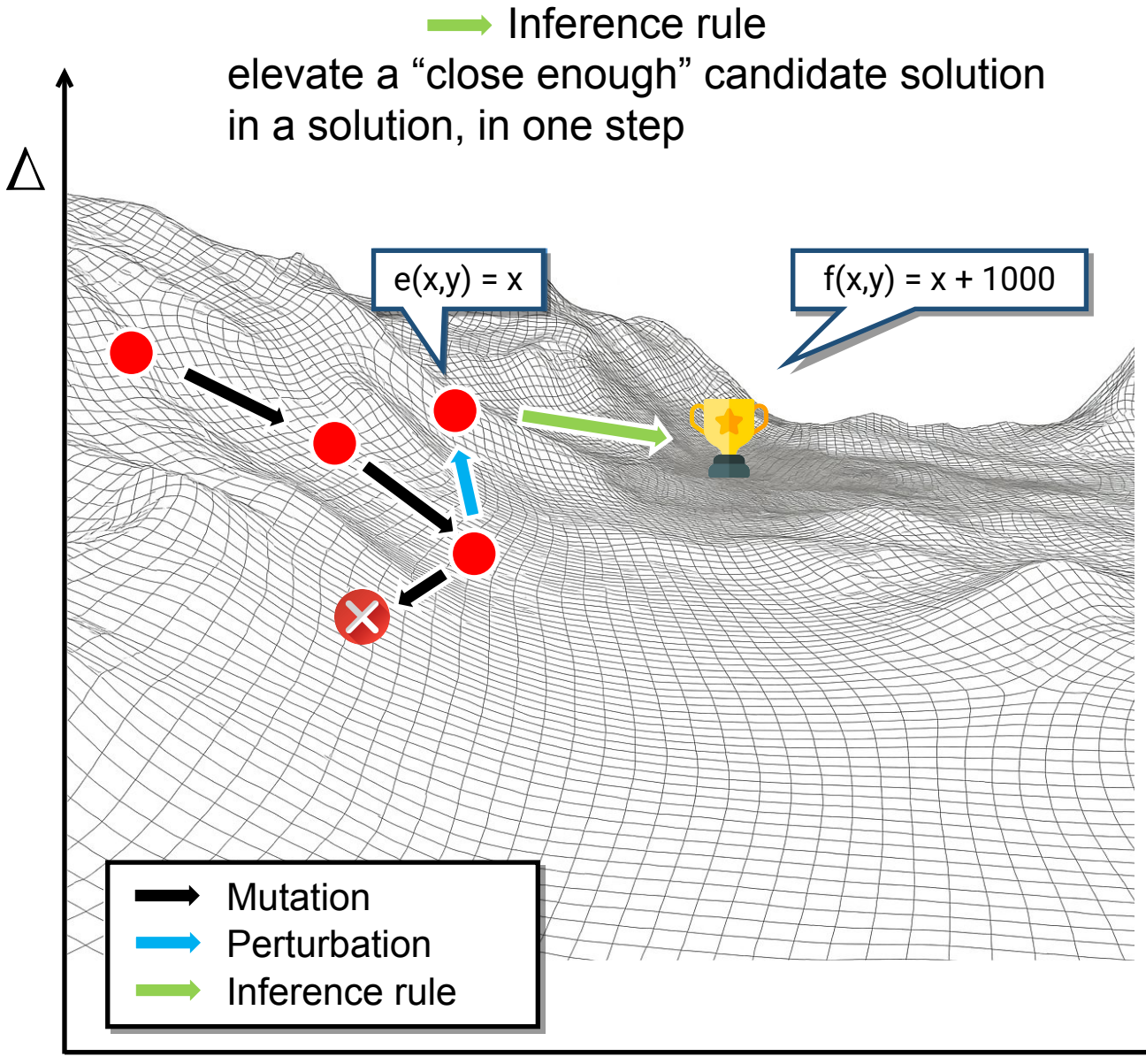
In the last decade, *white-box deobfuscation* techniques, lifting the latest methods of static program analysis, have proved highly effective against many standard obfuscation schemes like opaque predicates [7, 54, 60] or virtualization [32, 51, 54]. However, several obfuscation methods significantly impair white-box analysis. Dedicated protections, including Mixed Boolean Arithmetic (MBA) expressions [63], covert channels [55] or path-oriented protections [3, 44, 45, 59], have emerged, many of which increase the syntactic complexity of the original code to break white-box approaches.

The promises of black-box deobfuscation. New deobfuscation methods, based on program synthesis, have emerged to simplify heavily obfuscated local code snippets [12, 39]. These methods rely only on *input-output* (I/O) *observations* to infer the code snippet behavior. Through synthesis, they explore a *space of candidate expressions* generated by an *inference grammar*, seeking an expression that mimics the I/O observations. Unlike white-box approaches, black-box ones are immune to syntactic complexity, but the analyzed code must be semantically simple to synthesize it. This explains why black-box methods primarily focused on virtualized code: handlers

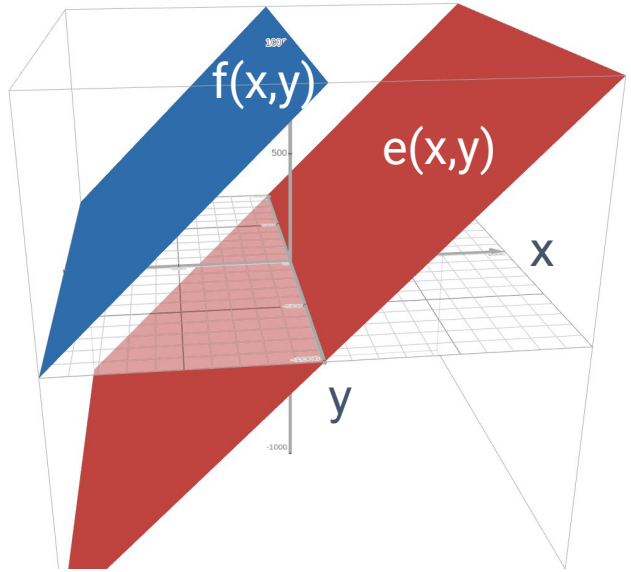
Search Modulo Inference Rules (SMIR)



Search Modulo Inference Rules (SMIR)



Inference rule for +



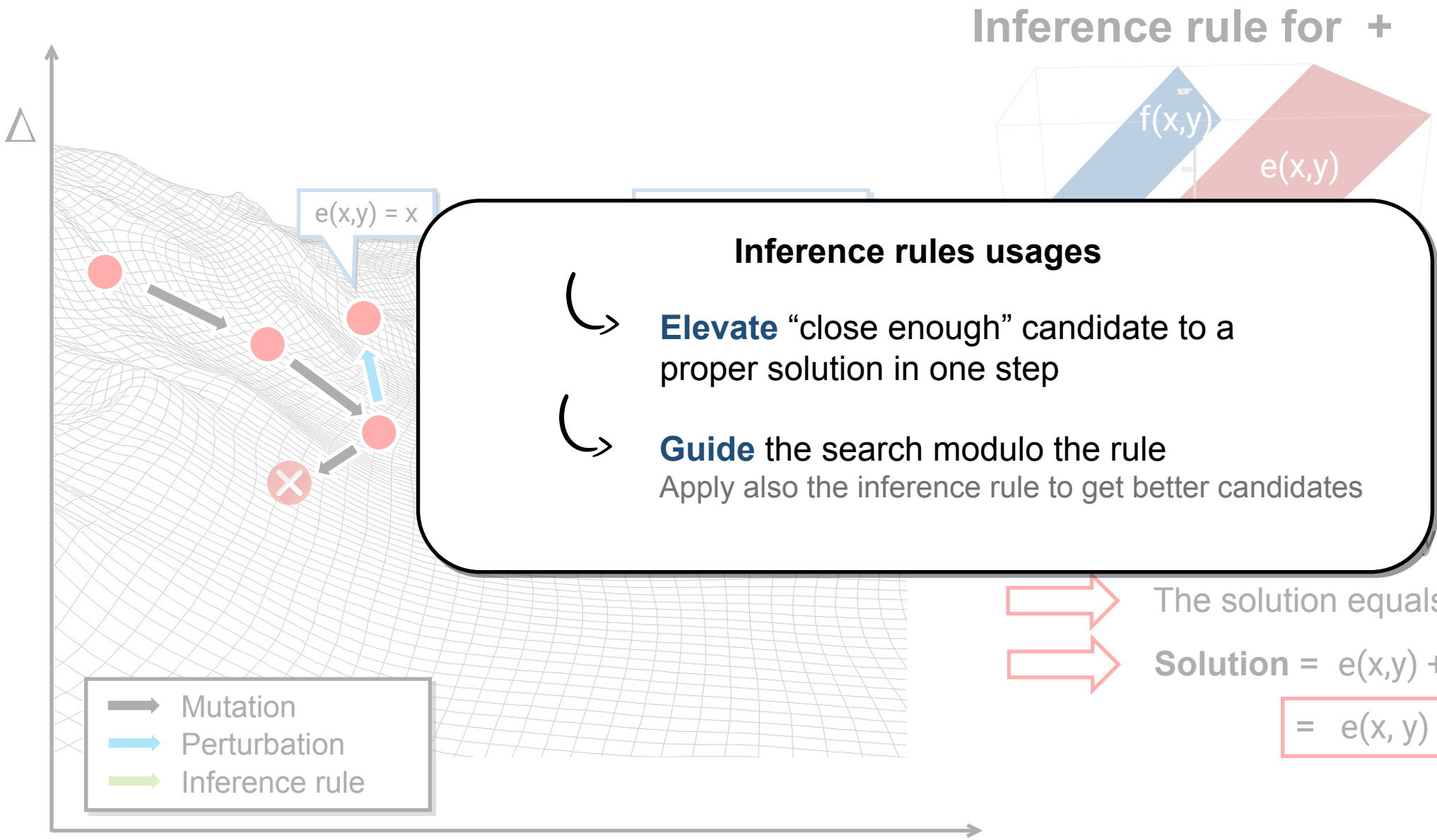
The variance of f(x,y) - e(x,y) equals 0

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

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

= e(x, y) + 1000

Search Modulo Inference Rules (SMIR)



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
- 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 *tzon
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

- cvc4/5
- DryadSynth
- Syntia
- Xyntia 2021

Xyntia 2025

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

- cvc4/5
- DryadSynth
- Syntia
- Xyntia 2021

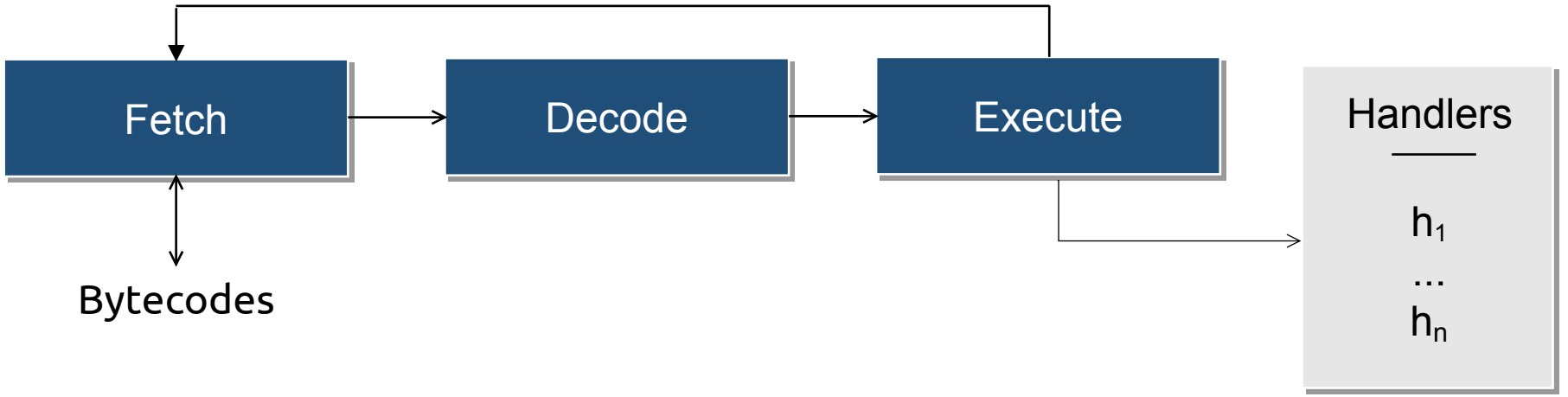
Xyntia 2025

500ms

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

Use case: VM handlers

An example with Tigress



Problem: Handlers can be obfuscated
> How to deobfuscated them ?

Use case: VM handlers

An example with Tigress

```
[0x1578]
  ; DATA XREF from sym.foo @ 0x120f(r)
128: fcn.00001578 ();
afv: vars(2:sp[0x2c8..0x2d0])
endbr64
mov rax, qword [var_2d0h]
add rax, 8
mov qword [var_2d0h], rax
mov rax, qword [var_2c8h]
mov edx, dword [rax]
mov rax, qword [var_2c8h]
sub rax, 0x10
mov eax, dword [rax]
xor eax, edx
not eax
mov ecx, eax
mov rax, qword [var_2c8h]
sub rax, 0x10
mov edx, dword [rax]
mov rax, qword [var_2c8h]
mov eax, dword [rax]
or eax, edx
add eax, eax
lea edx, [rcx + rax]
mov rax, qword [var_2c8h]
sub rax, 0x10
add edx, 1
mov dword [rax], edx
mov rax, qword [var_2c8h]
sub rax, 0x10
mov qword [var_2c8h], rax
mov rax, qword [var_2d0h]
mov rax, qword [rax]
; sym.foo+0x1df
jmp 0x1328
```

Obfuscated with MBA

Xyntia Script

starting from 0x1578

explore all

hook 0x15f3 with
  sample 100
  halt
end

```
→ xyntia xyntia -bin binary.exe -config script
  expression: (mem_1<32> + mem_2<32>)
  simplified: (mem_1<32> + mem_2<32>)
  smtlib: (bvadd mem_1 mem_2)
  output: outmem1<32>
  success: yes
  synthesis time: 0.000251
  equiv: yes
```

Ok, so it's only an addition handler 😎

Conclusion

- Xyntia is a new framework for deobfuscation **It is under active research**
- Interested by Xyntia and black-box deobfuscation ?

Read our research papers

> More details & lot of evaluations

Session 10A: Crypto, Symbols and Obfuscation CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

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

Grégoire Menguy Sébastien Bardin

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

Vidal Attias, Université Paris-Saclay, CEA, List, Palaiseau, France, vidal.attias@cea.fr
Nicolas Bellec, Université Paris-Saclay, CEA, List, Palaiseau, France, nicolas.bellec@cea.fr
Grégoire Menguy, Université Paris-Saclay, CEA, List, Palaiseau, France, gregoire.menguy@cea.fr
Sébastien Bardin, Université Paris-Saclay, CEA, List, Palaiseau, France, sebastien.bardin@cea.fr
Jean-Yves Marion, Université de Lorraine, CNRS, LORIA, Nancy, France, jean-yves.marion@loria.fr

Abstract

Code obfuscation aims to protect programs from reverse engineering, with applications ranging from intellectual property protection to malware hardening. Recent works on black-box analyses propose to leverage program synthesis in order to infer the semantics of highly obfuscated code blocks. Being fully *black-box*, these approaches are immune to *syntactic* complexity and can thus bypass standard obfuscation mechanisms. Yet, they are restricted by their synthesis capabilities and can only be applied to *semantically* simple code blocks. It explains why they have mainly been used on virtual machine handlers, where behaviors are usually simple enough. Applying black-box deobfuscation at scale beyond virtualization is still an open problem, notably because black-box methods cannot synthesize complex behaviors involving, for example, arbitrary constant values or affine or polynomial relations over mixed-boolean-arithmetic expressions. In this article, we show how to combine *search-based program synthesis* with *local inference rules*, resulting in a new method named *Search Modulo Inference Rules* (SMR) that boosts search-based program synthesis while keeping its generality and flexibility. We instantiate SMR with inference rules for hard synthesis problems like arbitrary constant values and affine or polynomial relations over mixed boolean expressions, yielding the new black-box deobfuscation tool: XSMR. Experiments demonstrate that XSMR significantly outperforms prior black-box deobfuscators, synthesizing overall 76% and 84% of the expressions from our real-world obfuscated and non-obfuscated benchmarks where prior works recover 63% and 55%, together with 2 to 3 times less false positive and slightly improved compression rate.

CCS Concepts

• **Security and privacy** → **Software reverse engineering**; • **Computing methodologies** → **Randomized search**.

Keywords

Deobfuscation, reverse engineering, program synthesis

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765134

ACM Reference Format:
Vidal Attias, Nicolas Bellec, Grégoire Menguy, Sébastien Bardin, and Jean-Yves Marion. 2025. Augmenting Search-based Program Synthesis with Local Inference Rules to Improve Black-box Deobfuscation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765134

1 Introduction

Obfuscation [18, 19] aims to protect software from reverse engineering. It translates a program P into a functionally equivalent program P_{obf} harder to analyze. As cryptography has not yet provided a bullet-proof solution for this problem, the obfuscation community mainly focuses on program-analysis-based techniques [19], leading to a cat-and-mouse game between defenders and attackers.

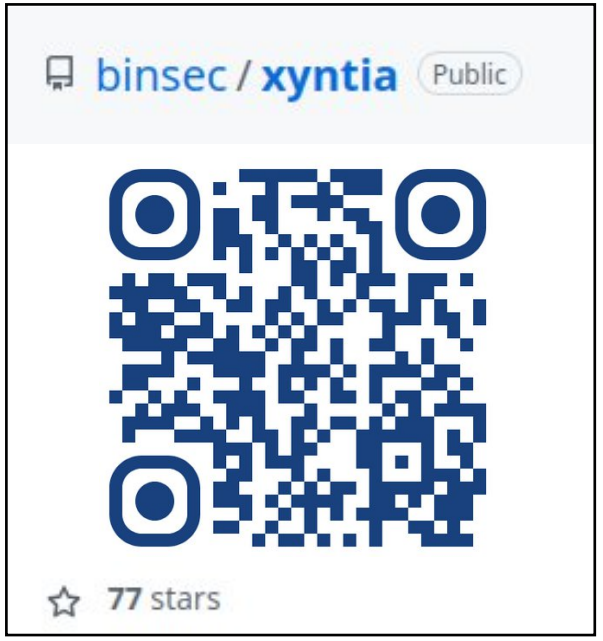
While obfuscation is used to protect *Intellectual Property* and other valuable software assets, it is also used to protect malware. Thus, automated *deobfuscation* methods [12, 21, 39, 50, 51] have been proposed to cope with the quick advances in obfuscation. Given an obfuscated program P_{obf}, the goal here is to simplify it into a simpler yet functionally equivalent program $P^* -$ ideally P^* should be as simple as the original unprotected code P.

In the last decade, *white-box deobfuscation* techniques, lifting the latest methods of static program analysis, have proved highly effective against many standard obfuscation schemes like opaque predicates [7, 54, 60] or virtualization [32, 51, 54]. However, several obfuscation methods significantly impair white-box analysis. Dedicated protections, including Mixed Boolean Arithmetic (MBA) expressions [63], covert channels [55] or path-oriented protections [3, 44, 45, 59], have emerged, many of which increase the syntactic complexity of the original code to break white-box approaches.

The promises of black-box deobfuscation. New deobfuscation methods, based on program synthesis, have emerged to simplify heavily obfuscated local code snippets [12, 39]. These methods rely only on *input-output* (I/O) *observations* to infer the code snippet behavior. Through synthesis, they explore a *space of candidate expressions* generated by an *inference grammar*, seeking an expression that mimics the I/O observations. Unlike white-box approaches, black-box ones are immune to syntactic complexity, but the analyzed code must be semantically simple to synthesize it. This explains why black-box methods primarily focused on virtualized code: handlers

Try Xyntia yourself

> Give us feedback



Looking for PhD or internships?

> Send me an email

Contact information here:
https://gregoiremenguy.github.io/