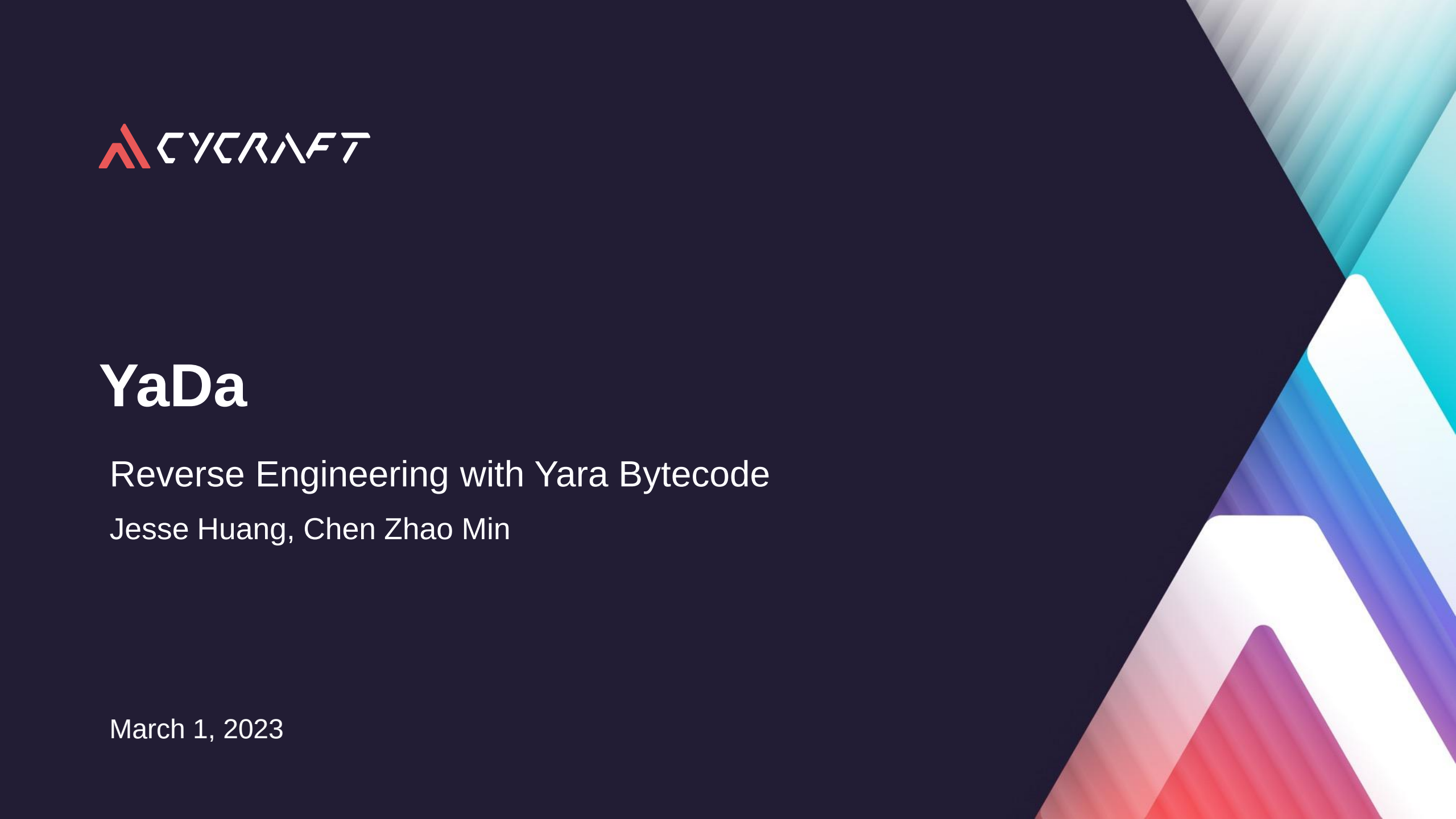


YaDa

Reverse Engineering with Yara Bytecode

Jesse Huang, Chen Zhao Min

March 1, 2023

Jesse Huang

- ~~ラーメンマニア~~
- Compiler Engineer @ SiFive
- Ex-Security Researcher Intern @ CyCraft
- CTF Player: 10sec, TSJ, TWN48, …
- This work is launched and basically done while I was an intern at CyCraft (2020~2021), but I was too busy to clean it up after I went to grad school...

Disclaimer: This work is not related to Jesse's work at SiFive

ZHAO MIN CHEN

- Security Researcher @ CyCraft
- CTF Player: TWN48, Balsn, w33d …
- Presentation: USENIX 2024 Poster, AVTokyo, …
- GitHub: asef18766

Motivations

- Yara is often used, but we didn’t quite understand how it works
- Our motivation is to reveal the internals of Yara
  - Understand how could it be that fast
  - See if there are components that could be utilized or improved
  - See if yara rules could be decompiled!

Outline

- Yara Introduction
- YaDa Introduction
- YaDa implementation
  - Rule Decompilation
    - Rule Bytecode Instruction Set
    - Rule decompile algorithm
  - Pattern Decompilation
    - Regex Bytecode Instruction Set
    - Version Dependent Decompilation
      - v3.4.0/v3.9.0 hex pattern/regex decompilation
- YaDa Evaluation

Yara Introduction

Yara

- A multithreaded, multi-pattern matching tool
- De facto industry standard for identifying certain characteristics of malware

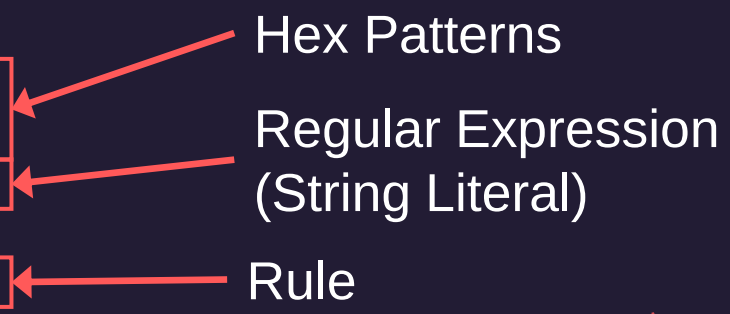
{ } Who's using YARA

- ActiveCanopy
- Adlice
- AlienVault
- Avast
- BAE Systems
- Bayshore Networks, Inc.
- BinaryAlert
- Blueliv
- Cisco Talos Intelligence Group
- Claroty
- Cloudina Security
- Cofense
- Conix
- CounterCraft
- Cuckoo Sandbox
- Cyber Triage
- Digita Security
- Dragos Platform
- Dtex Systems
- ESET
- ESTSecurity
- Fidelis XPS
- FireEye, Inc.
- Forcepoint
- Fox-IT
- FSF
- Guidance Software
- Heroku
- Hornetsecurity
- ICS Defense
- InQuest
- Joe Security
- Kaspersky Lab
- KnowBe4
- Koodous
- Laika BOSS
- Lastline, Inc.
- libguestfs
- LimaCharlie
- Malpedia
- Malwation
- McAfee Advanced Threat Defense

Yara Syntax

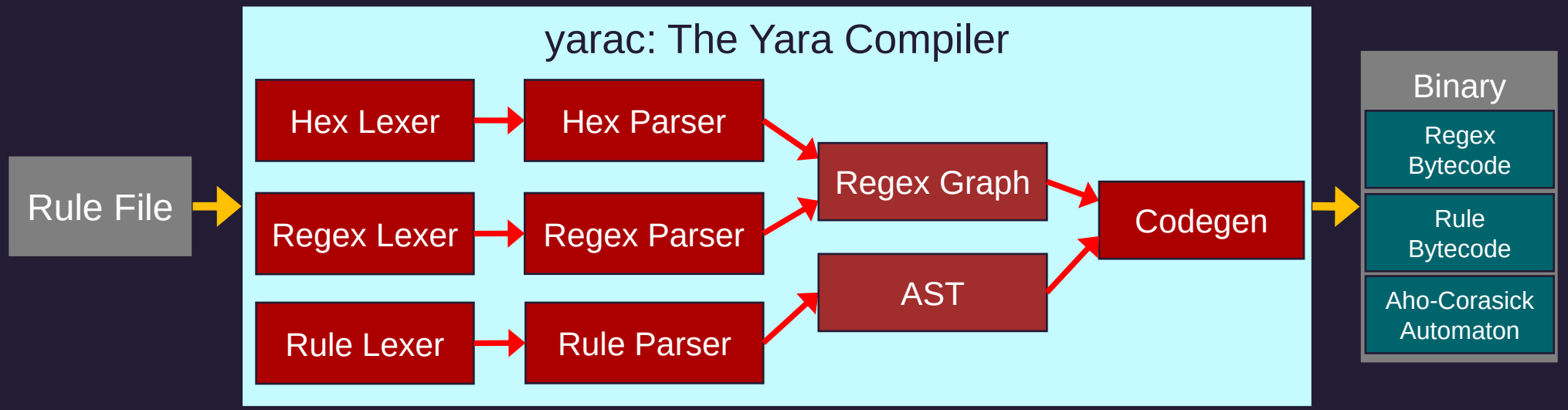
- A language for describing patterns and defining how to match the patterns
- Support **regexes**, **literals** and **hex patterns**

```
rule silent_banker : banker
{
   meta:
      description = "This is just an example"
      threat_level = 3
      in_the_wild = true
   strings:
      $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
      $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
      $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
   condition:
      $a or $b or $c
}
```



Rule File

- An Yara rule file consists of one or more Yara rules
- Could be **compiled into binary** for speed up the scanning process



Yara Binary

- Unstable, varies from version to version
- Header
  - Yara Version
  - Segment information
- Code Segment
  - Rule bytecode
  - Regex bytecode
- Rules Segment
  - Rules
- String Table
  - String (which really means pattern)

Header
Automata Segment
Code Segment
Rules Segment
External Segment
Relocation Table
String Table

Rule Bytecode

- Yara rule is compiled into **bytecode**
- The instruction set is unstable

```
rule Rule02 {
strings:
    $a1  = "str1"
    $a2  = "str2"
    $a3  = /asdf.*zxcv{1,8}/
    $a4  = { aa bb cc dd [1-20] ee ff }
    $a5  = { 00 01 02 03 04 05 06 }
    $b4  = { 00 05 06 }
    $b5  = { 00 [1-5] 06 07 08 09 0a [1-10] 0b 0c }
    $mz  = "MZ"
condition:
    (all of them) or ((3 of them) and $mz at 0)
}
```

```
OP_INIT_RULE ( 0x6C3C )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0x6E40  $a1 )
OP_PUSH ( 0x7370  $a2 )
OP_PUSH ( 0x78A0  $a3 )
OP_PUSH ( 0x7DD0  $a4 )
OP_PUSH ( 0x8300  $a5 )
OP_PUSH ( 0x8830  $b1 )
OP_PUSH ( 0x8D60  $b2 )
OP_PUSH ( 0x9290  $b3 )
OP_PUSH ( 0x97C0  $b4 )
OP_PUSH ( 0x9CF0  $b5 )
OP_PUSH ( 0xA220  $b6 )
OP_PUSH ( 0xA750  $b7 )
OP_OF
OP_JTRUE ( 0x6919 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
...
```

Regex Bytecode

- Regular expressions are compiled into another set of **bytecode** and an **Aho-Corasick(AC) automata**
- The bytecode instruction set is based on Regular Expression Matching: the Virtual Machine Approach
- Likewise, it's unstable

```
$a3  = /asdf.*zxcv{1,8}/
```

```
RE_OPCODE_LITERAL ( 0x61 )
RE_OPCODE_LITERAL ( 0x73 )
RE_OPCODE_LITERAL ( 0x64 )
RE_OPCODE_LITERAL ( 0x66 )
RE_OPCODE_SPLIT_A ( 0x7 0x0 )
RE_OPCODE_ANY_EXCEPT_NEW_LINE
RE_OPCODE_JUMP ( -0x4 )
RE_OPCODE_LITERAL ( 0x7a )
RE_OPCODE_LITERAL ( 0x78 )
RE_OPCODE_LITERAL ( 0x63 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_PUSH ( 0x6 )
RE_OPCODE_SPLIT_A ( 0x8 0x0 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_JNZ ( -0x5 )
RE_OPCODE_POP
RE_OPCODE_SPLIT_A ( 0x5 0x0 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_MATCH
```

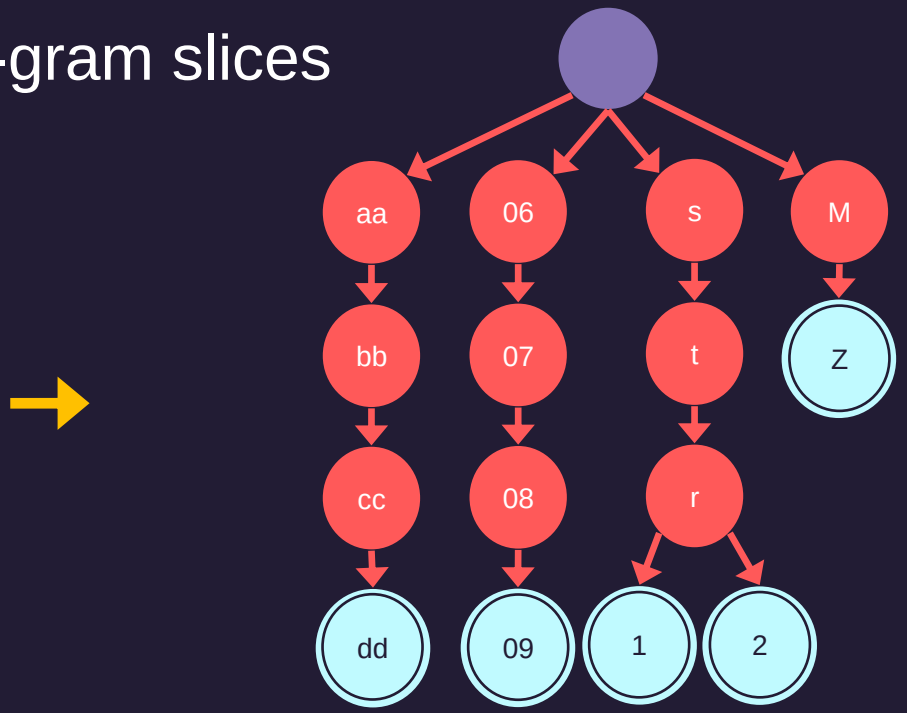
Regex Bytecode

- Regex matching is achieved by 2 stages
  1. 4-gram string literal matching
     - Full string matching is time-wasting
     - Use 4-gram to filter out most strings, then only verify potential matches
  2. Forward & backward verification (bytecode)

Regex Bytecode (4-gram)

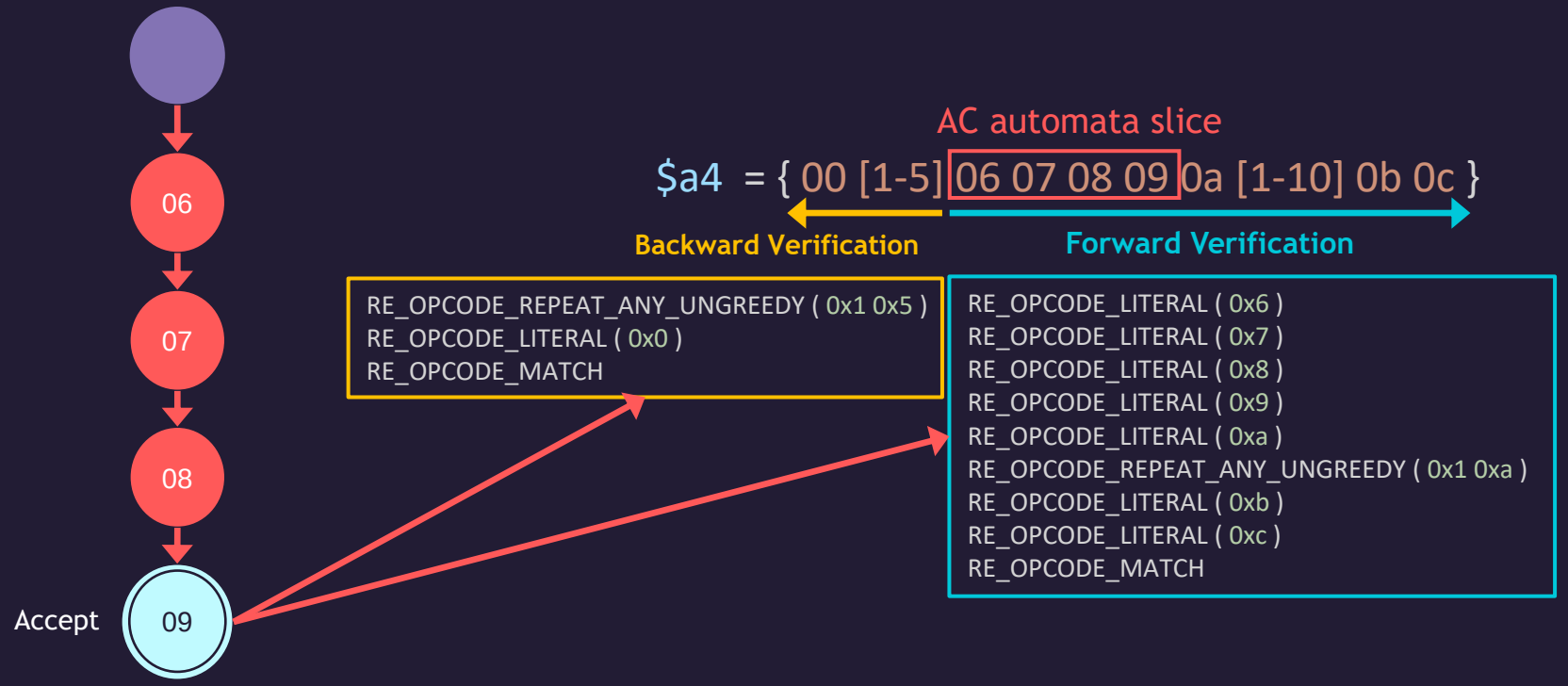
- String literal slices of patterns are combined into a single AC automata for fast scanning
  - Skip the strings that do not match any 4-gram slices
- Maximum length (depth) = 4

```
$a1 = "str1"
$a2 = "str2"
$a3 = { aa bb cc dd [1-20] ee ff }
$a4 = { 00 [1-5] 06 07 08 09 0a [1-10] 0b 0c }
$mz = "MZ"
```



Regex Bytecode (Forward & Backward)

- RE Bytecode is split into **forward** and **backward** for verification
- On matching, run the VM verification byte codes



Section Summary

- Yara separately compiles **rules** and **patterns** into bytecode
- Specifications of rule bytecode and pattern bytecode are unstable
- An **AC automata** is used to store **4-grams** extracted from patterns
- Pattern bytecode is executed to verify matches of the AC automata

YaDa Introduction

YaDa

https://github.com/jaidTw/YaDa

A decompiler to recover source from yara binaries

- Based on the work of jbgalet/yaradec
  - Only **disassembles** the rule bytecode
  - Can't decompile regex
- Support v3.4.0 & v3.9.0
- Capable to decompile >~90% of real world Yara rules

YaDa

- Some other API provided by the script
  - Output rule bytecode
  - Output pattern bytecode
  - Output rule AST (in JSON format)

YaDa

```
❯ ./yada.py ../test.yac
rule default:Rule01 {
        // ptr = 6c3c
        meta:
                author = “Jesse"
        strings:
        /*0x6e40*/      $a1 = "str1"
        /*0x7370*/      $a2 = "str2"
        /*0x78a0*/      $a3 = /asdf.*zxcv{1,8}/
        /*0x7dd0*/      $a4 = { aa bb cc dd [1-20] ee ff }
…
        /*0xa220*/      $b6 = { 11 22 3? 44 [1-5] 65 77 ?9 11 ?? 33 ?? 44 55 66 }
        /*0xa750*/      $b7 = { 00 ?3 2? ( 10 20 30 | 3? ) 1? [10-11] 78 }
        condition:
                (all of them) or ((3 of them) and (uint16(0) == 0x5a4d))
}
…
```

Steps to Decompile

1. Parse header
2. Relocate symbols
3. Parse rule & code segment
4. Decompile rules
5. Parse AC automaton (depends on version)
6. Decompile patterns

Rule Decompilation

```
OP_PUSH ( 0x0 )
OP_PUSH ( 0x64 )
OP_PUSH ( 0x12A4  $a )
OP_FOUND_IN
```

```
$a in 0 .. 0x64
```

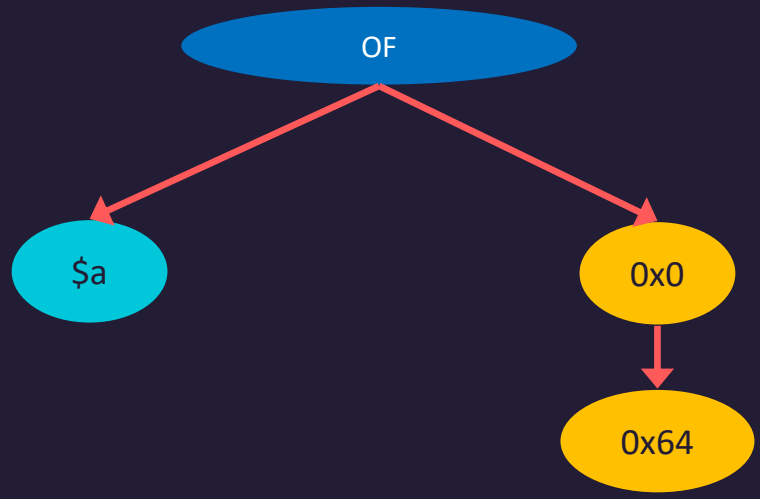
Rule Bytecode Instruction Set

- Stack Machine
- No optimization pass (good for decompilation)
- Most instructions have 0, 1 or 2 stack arguments
  - 0-arg : OP_PUSH, OP_INIT_RULE, OP_FILESIZE, …
  - 1–arg : OP_COUNT, OP_NOT, OP_INT8, …
  - 2-arg : OP_ADD, OP_SUB, OP_MUL, …
- Some Special Instructions
  - OP_FOUND_IN, OP_OF, OP_CALL, OP_OBJ_LOAD, …

Example: OP_FOUND_IN

```
OP_PUSH ( 0x0 )
OP_PUSH ( 0x64 )
OP_PUSH ( 0x12A4  $a )
OP_FOUND_IN
```

$a in 0 .. 0x64



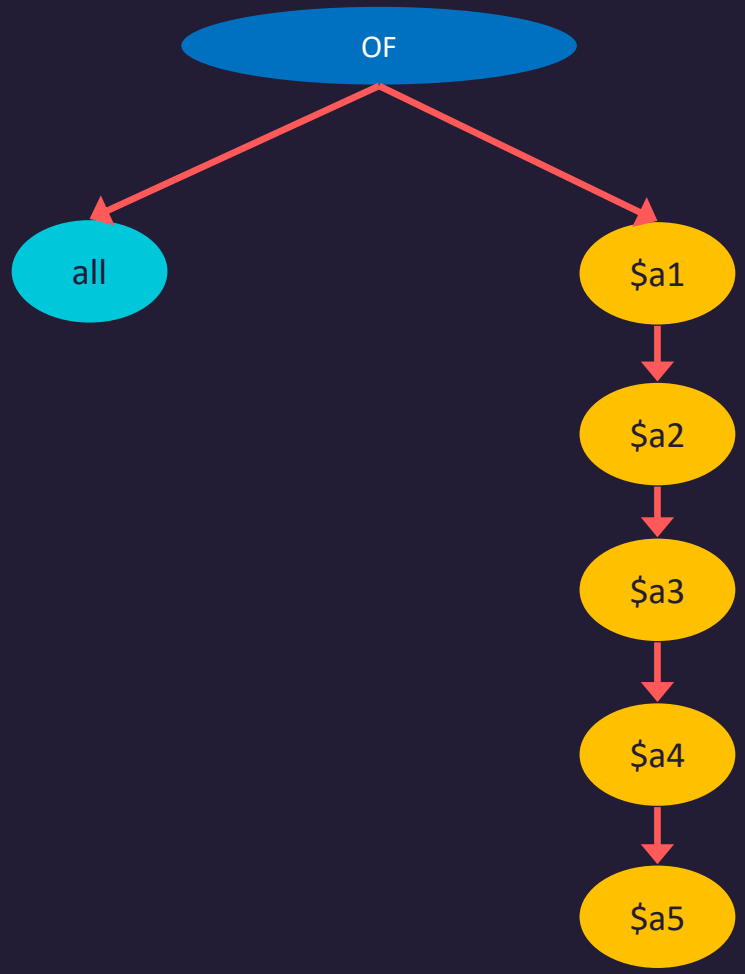
Example: OP_OF

```
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_OF
```

undefined = all

va_args terminated With **undefined**

all of $a1, $a2, $a3, $a4, $a5



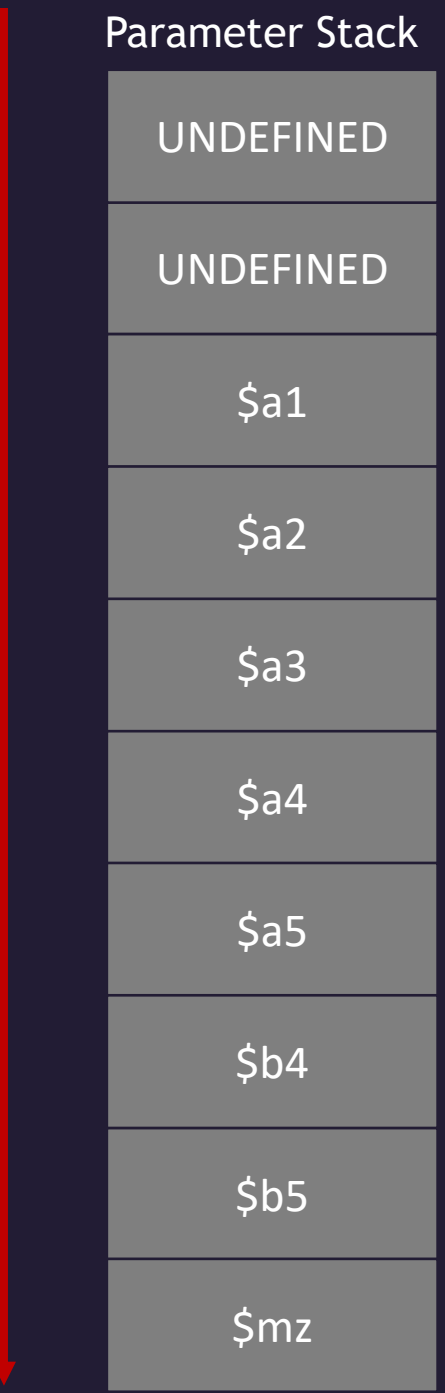
Decompiling Yara Rules

- Build an AST from the bytecode
  - Maintain a parameter stack while scanning through the code linearly
- Output rule according to the AST
  - Inorder traversal

```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack

```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```



Parameter Stack

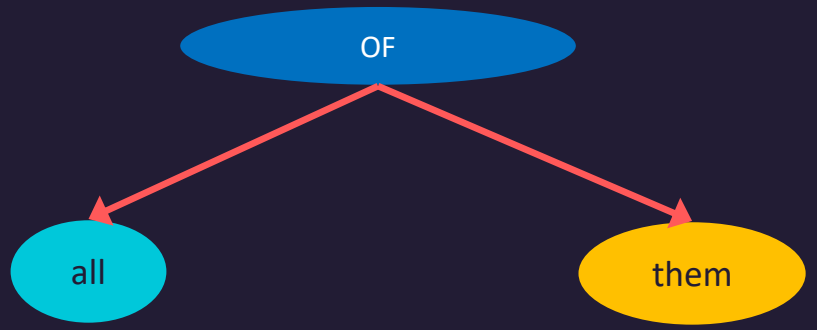
CYCRAFT

```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack

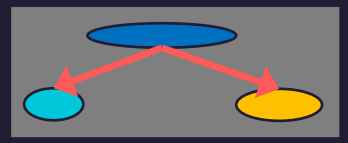
pop

UNDEFINED
UNDEFINED
$a1
$a2
$a3
$a4
$a5
$b4
$b5
$mz



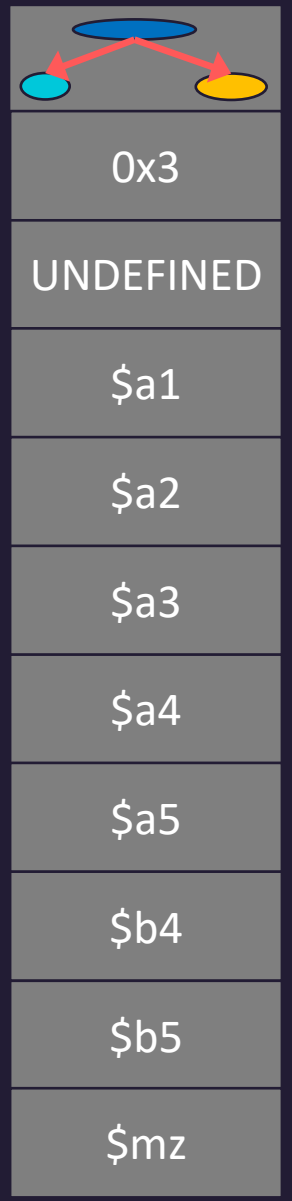
```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack



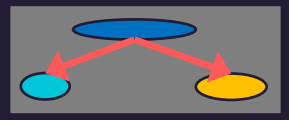
```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack



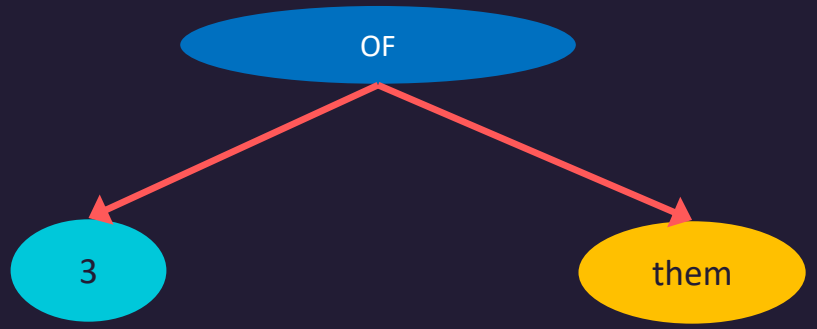
```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack



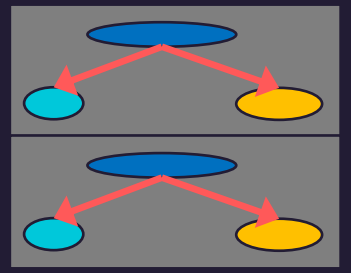
pop

0x3
UNDEFINED
$a1
$a2
$a3
$a4
$a5
$b4
$b5
$mz



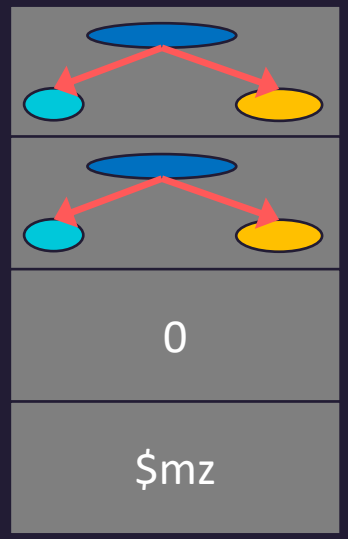
```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack



```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

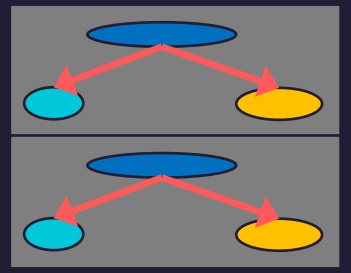
Parameter Stack



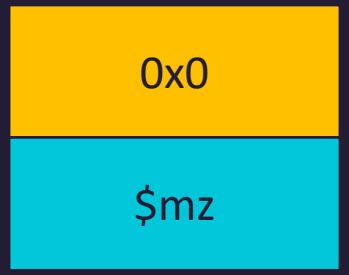
CYCRAFT

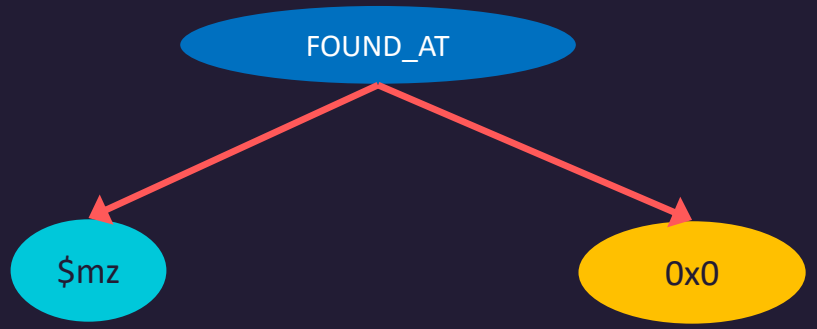
```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack



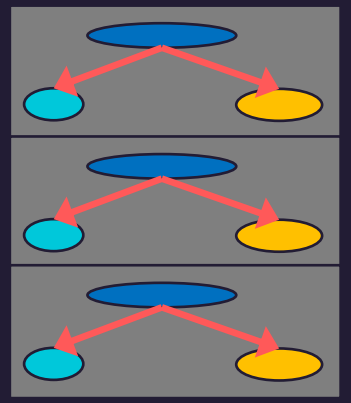
pop





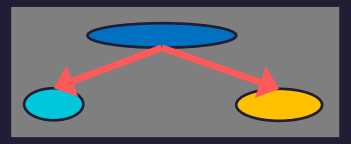
```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack

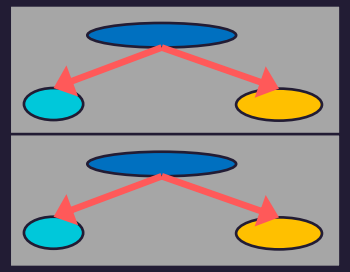


```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack



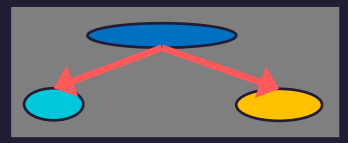
pop

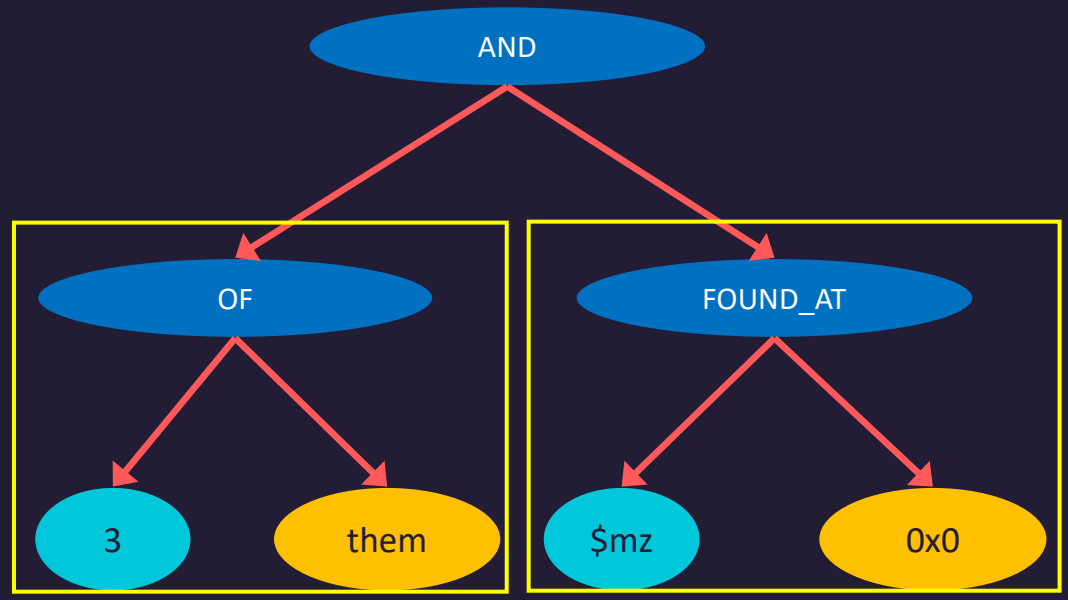


CYCRAFT

```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

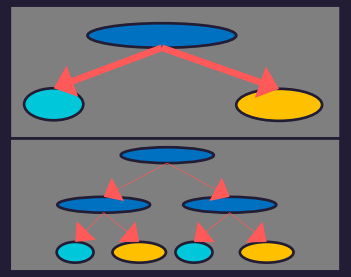
Parameter Stack

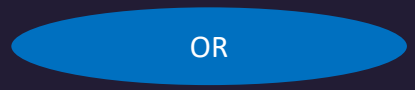




```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack

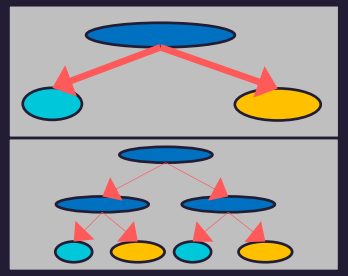


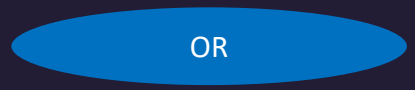


```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack

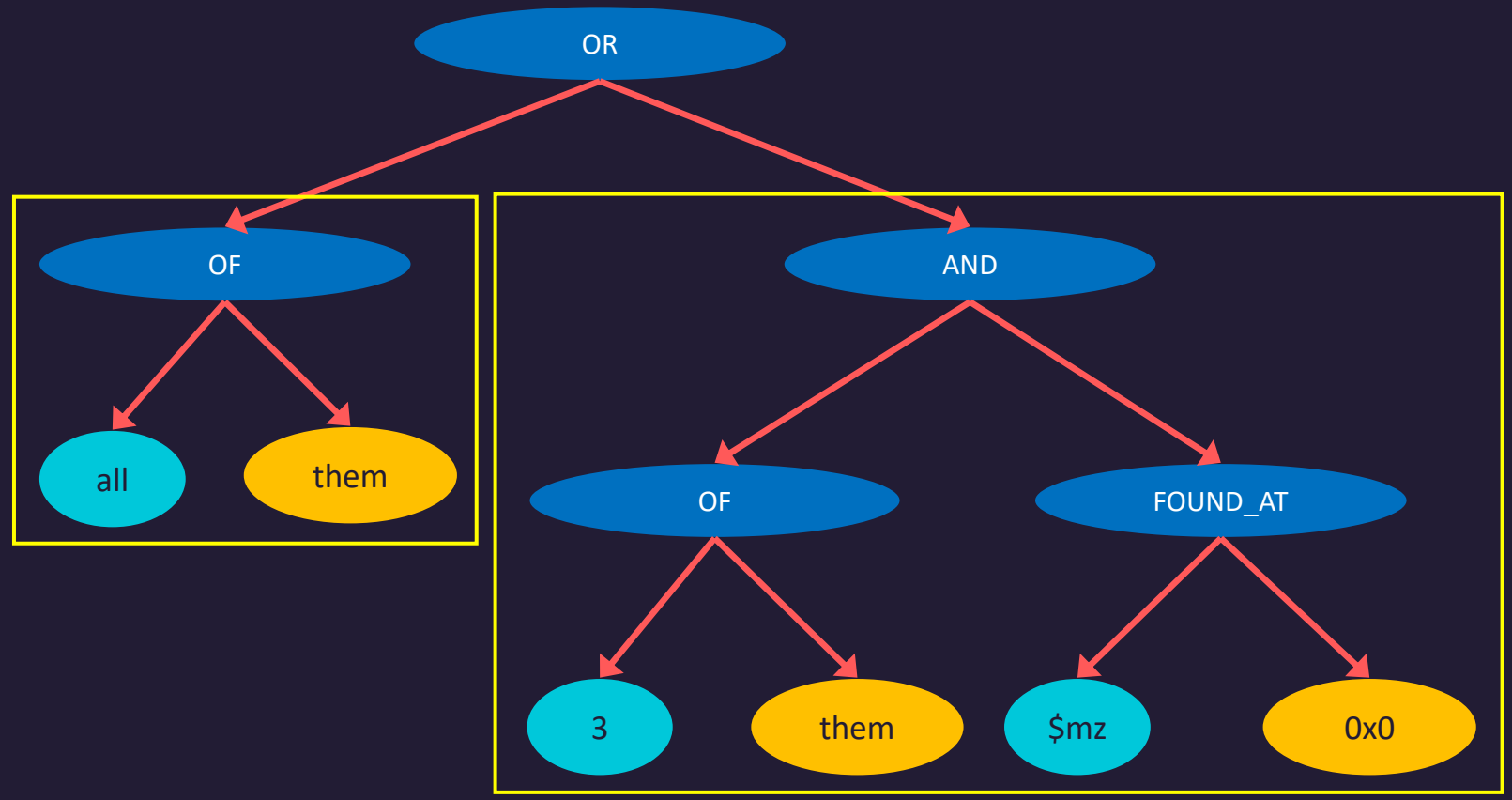
pop





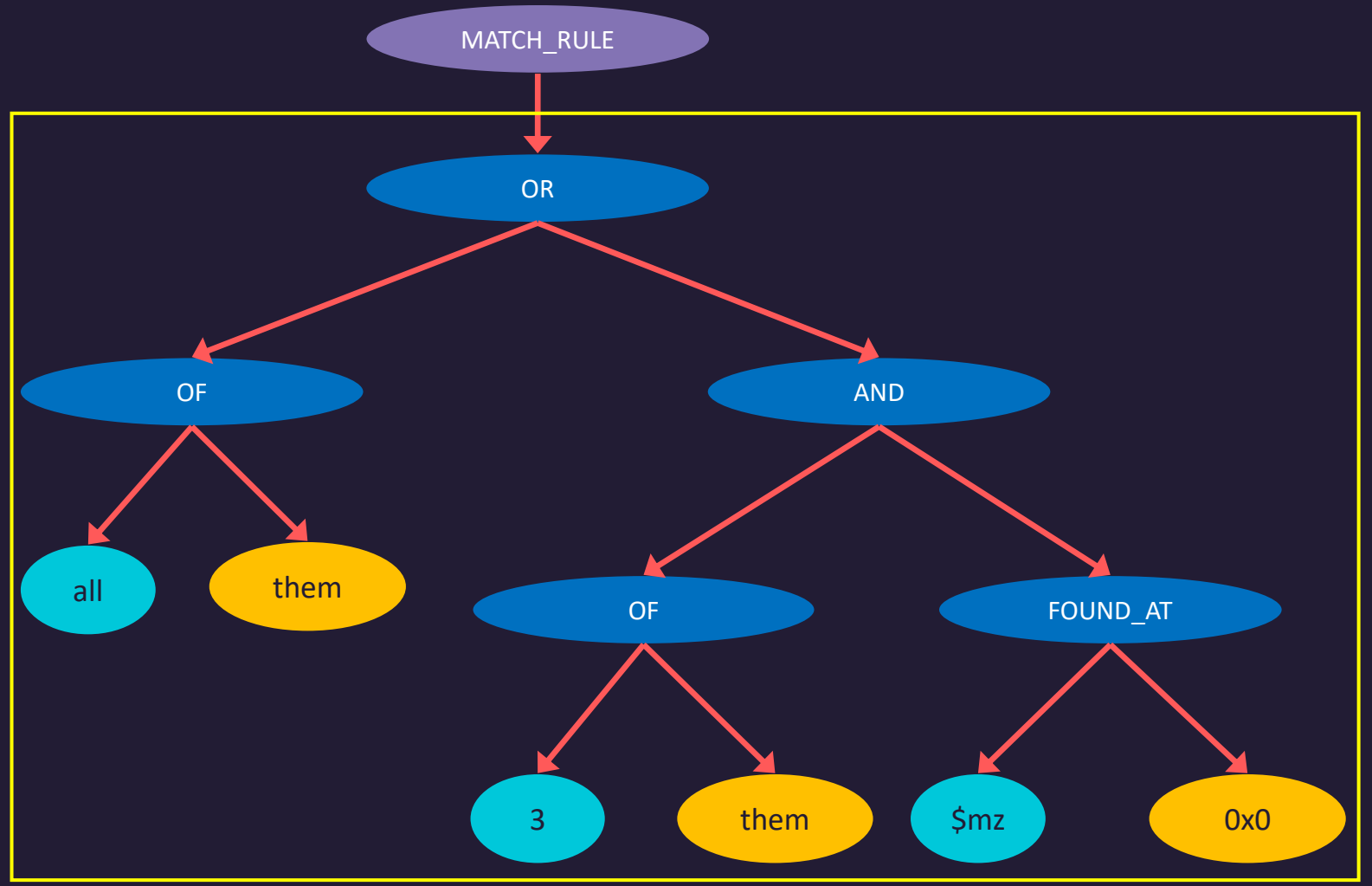
```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

Parameter Stack

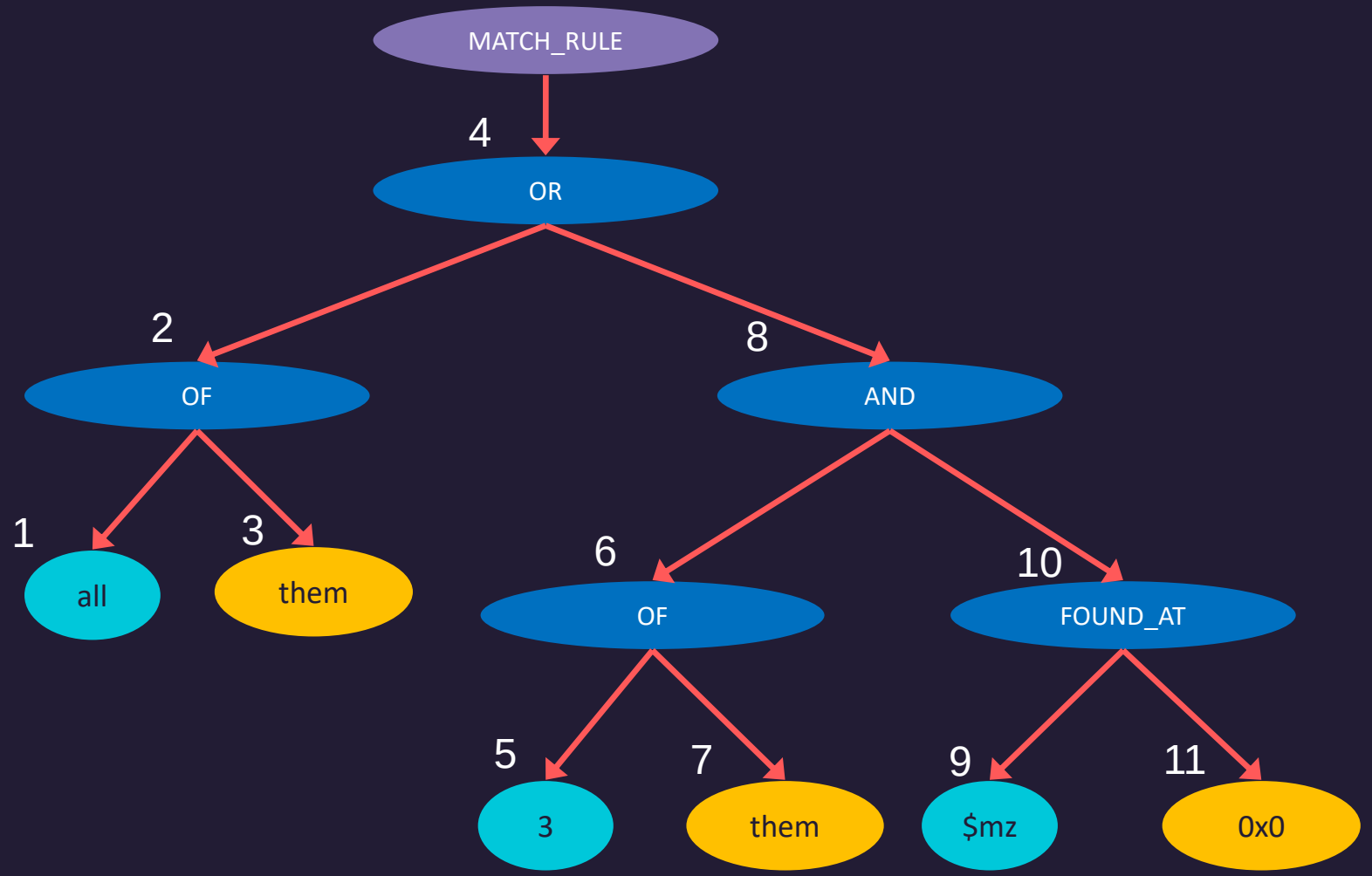


```
OP_INIT_RULE ( 0x6CE8 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JTRUE ( 0x6A08 )
OP_PUSH ( 0x3 )
OP_PUSH ( UNDEFINED )
OP_PUSH ( 0xB1B0  $a1 )
OP_PUSH ( 0xB6E0  $a2 )
OP_PUSH ( 0xBC10  $a3 )
OP_PUSH ( 0xC140  $a4 )
OP_PUSH ( 0xC670  $a5 )
OP_PUSH ( 0xCBA0  $b4 )
OP_PUSH ( 0xD0D0  $b5 )
OP_PUSH ( 0xD600  $mz )
OP_OF
OP_JFALSE ( 0x6A07 )
OP_PUSH ( 0x0 )
OP_PUSH ( 0xD600  $mz )
OP_FOUND_AT
OP_AND
OP_OR
OP_MATCH_RULE ( 0x6CE8 )
```

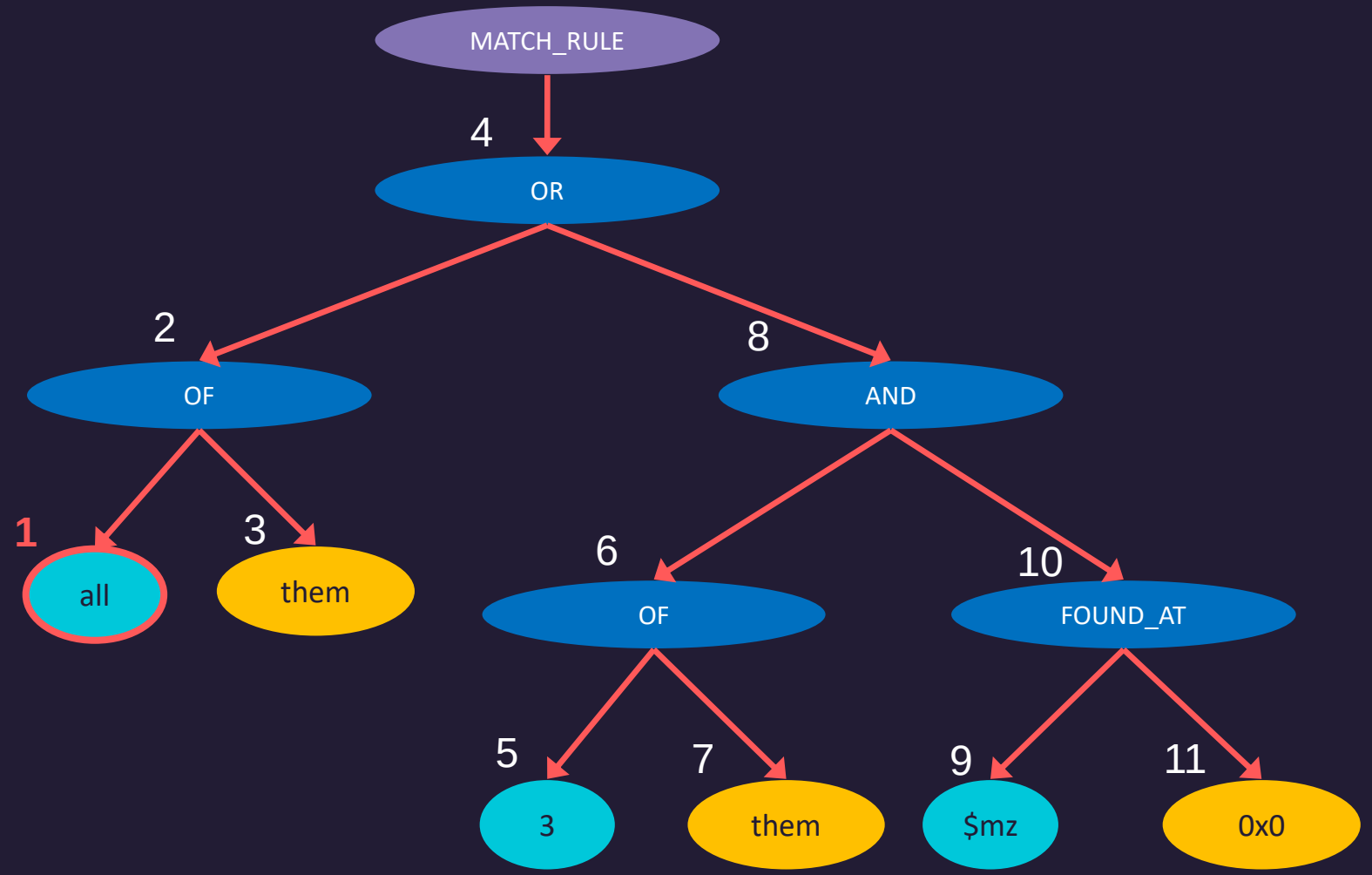
Parameter Stack



Output Rule

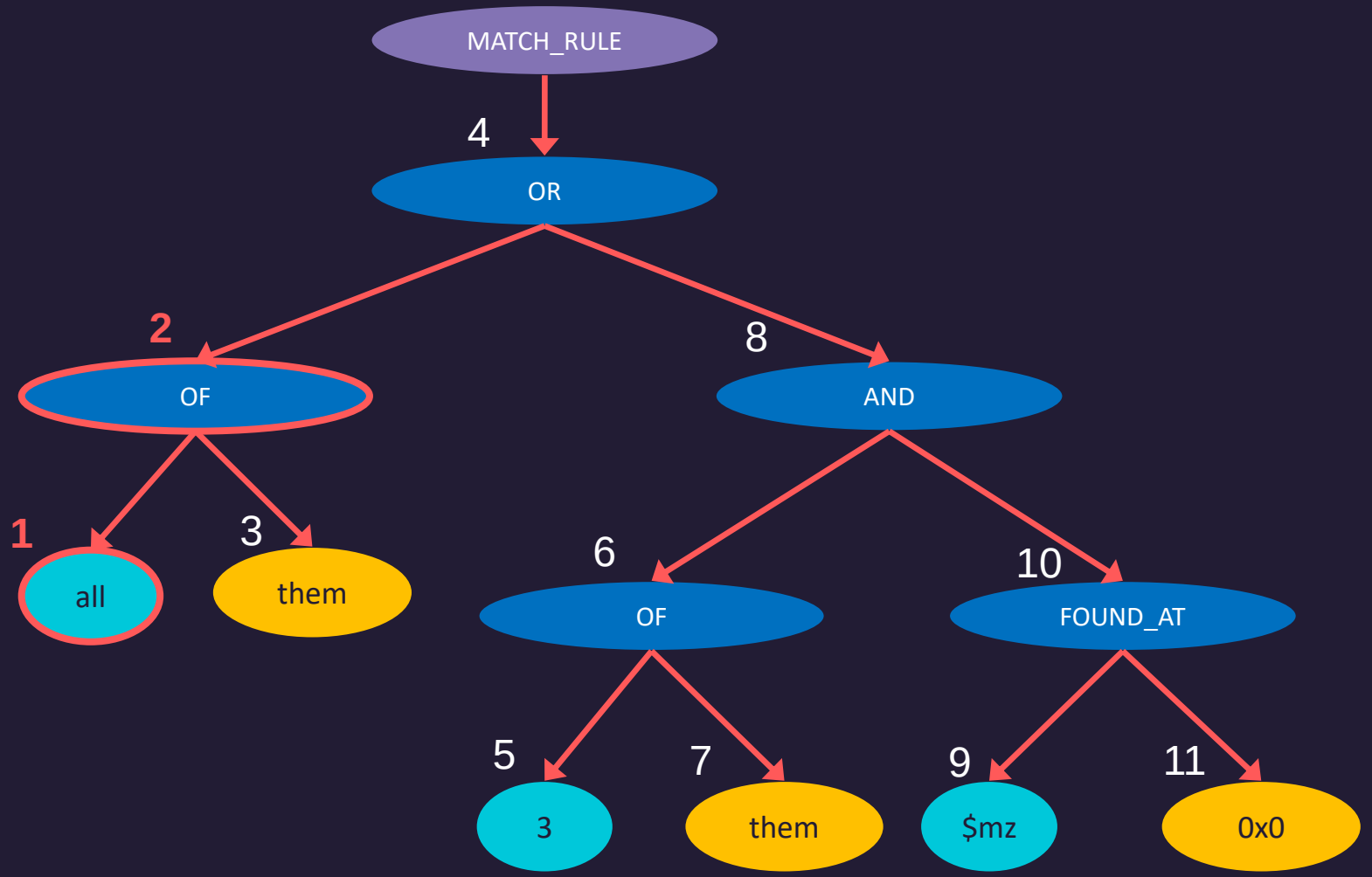


Output Rule



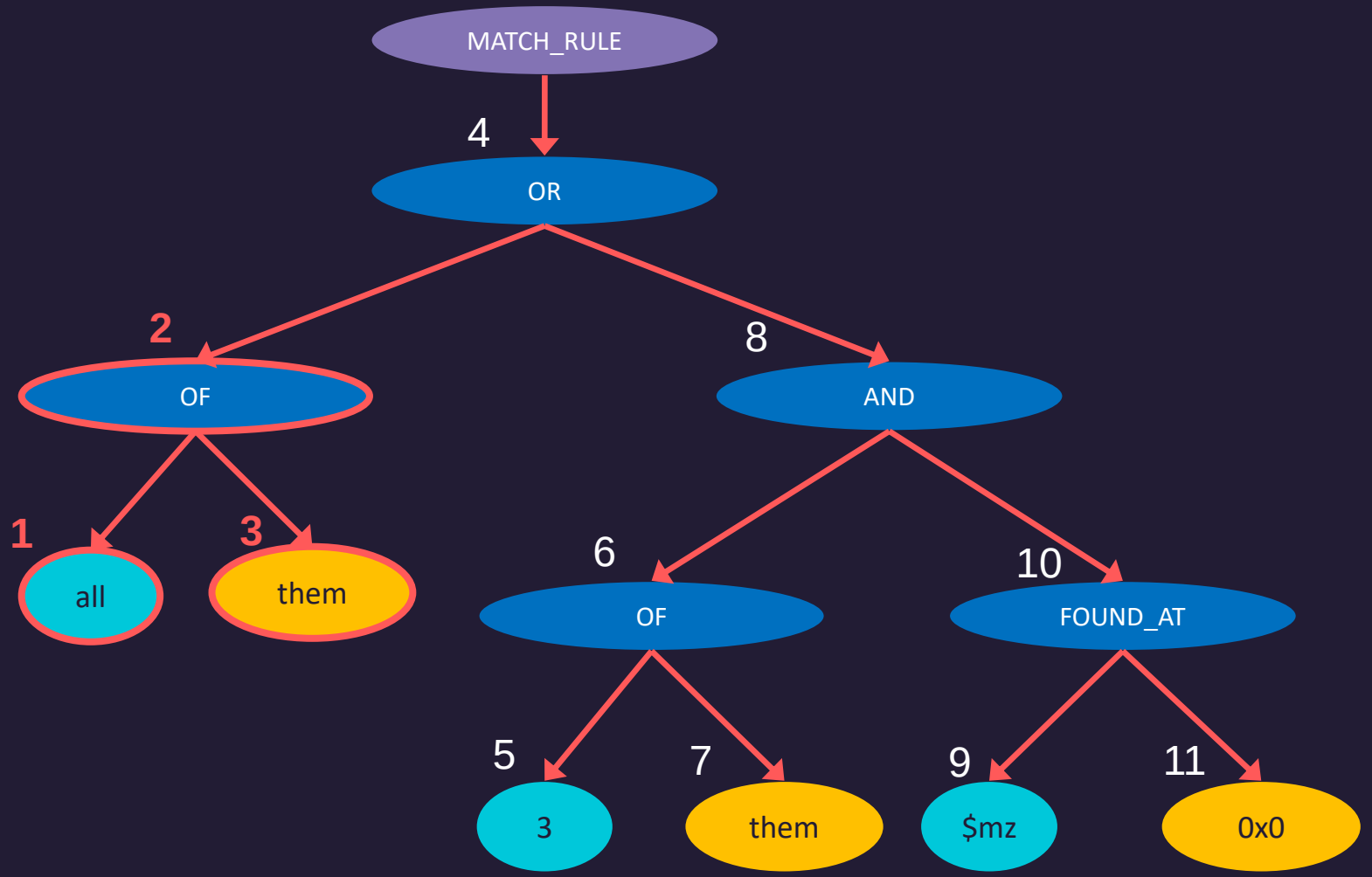
all

Output Rule



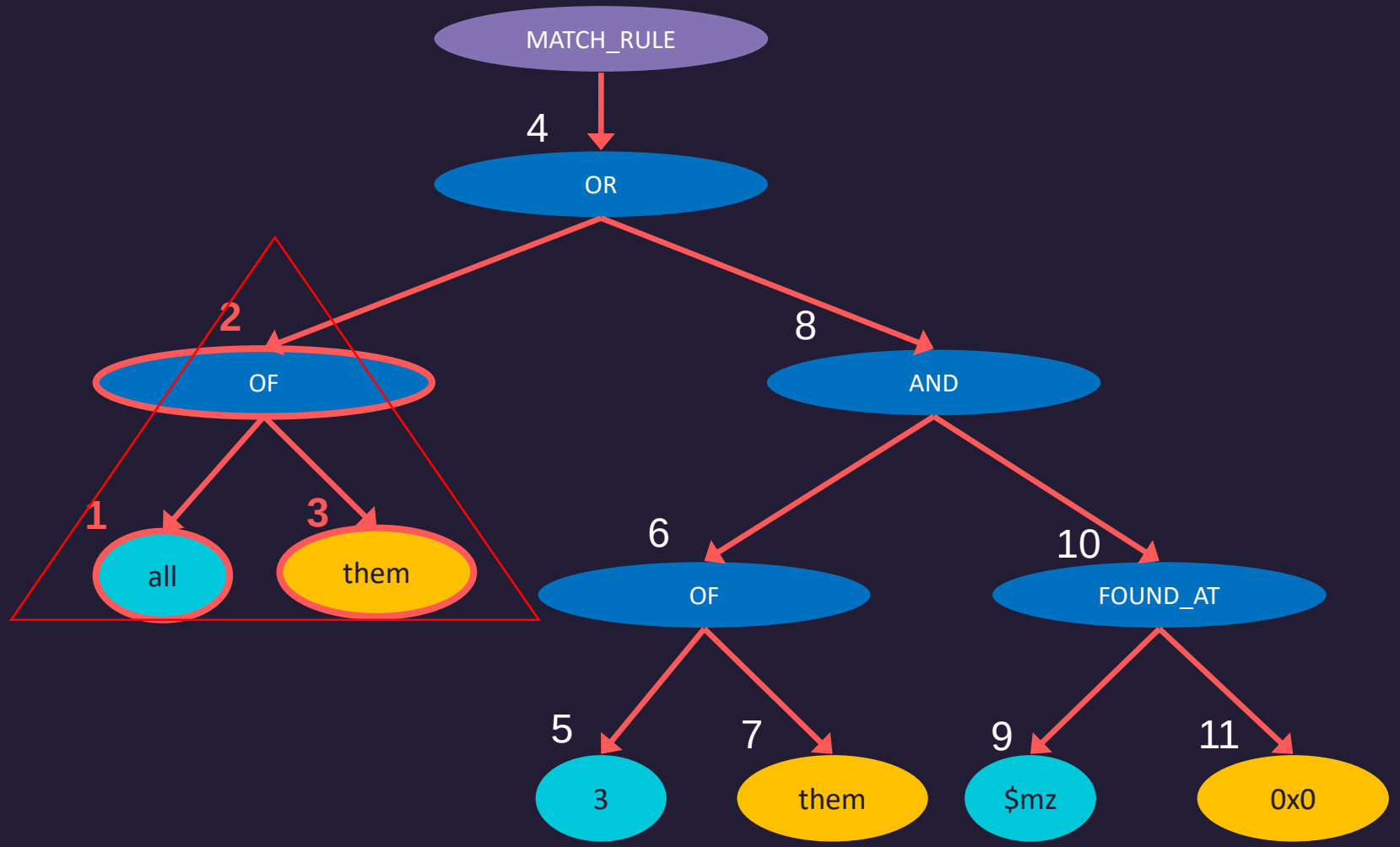
all of

Output Rule



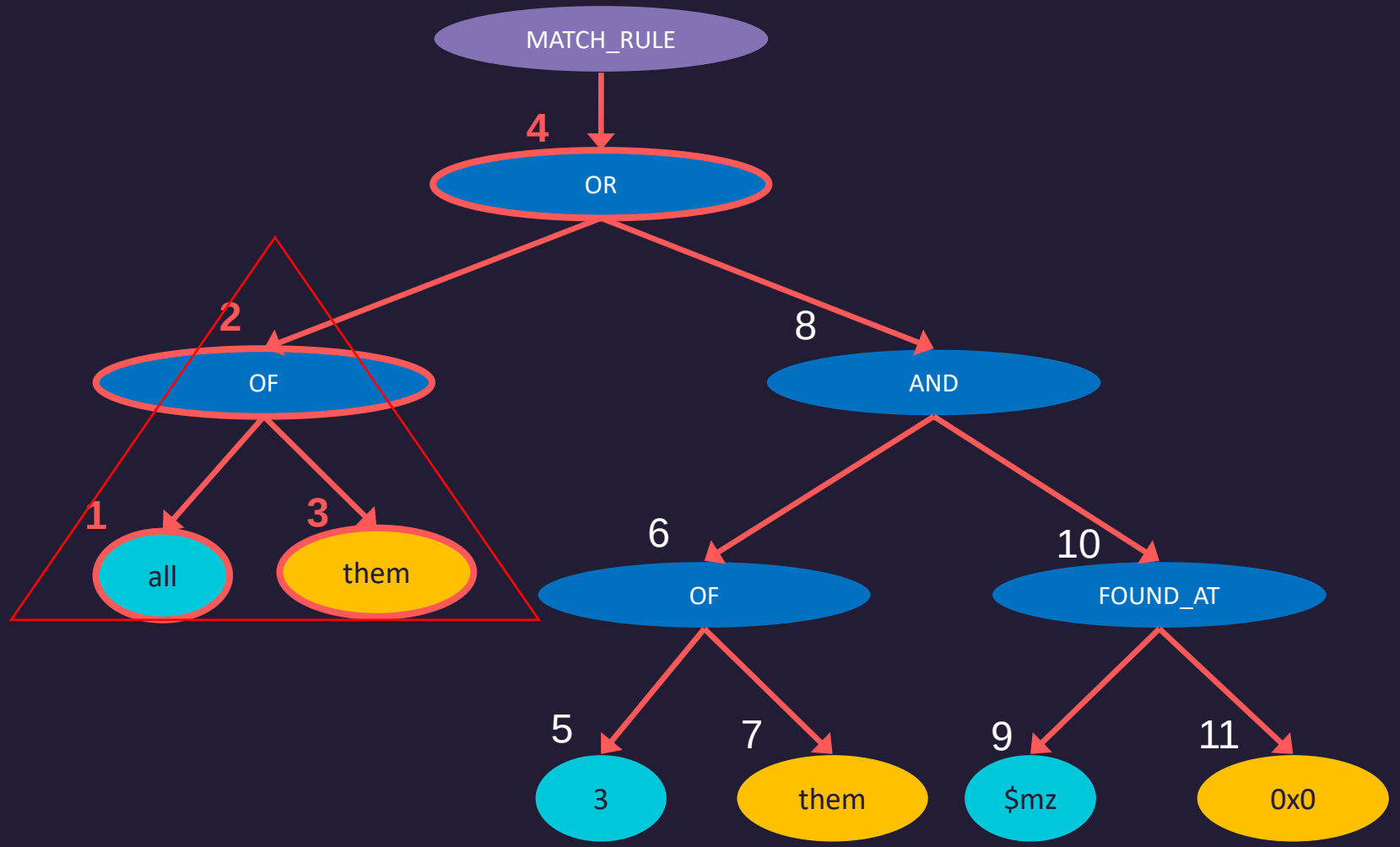
all of **them**

Output Rule



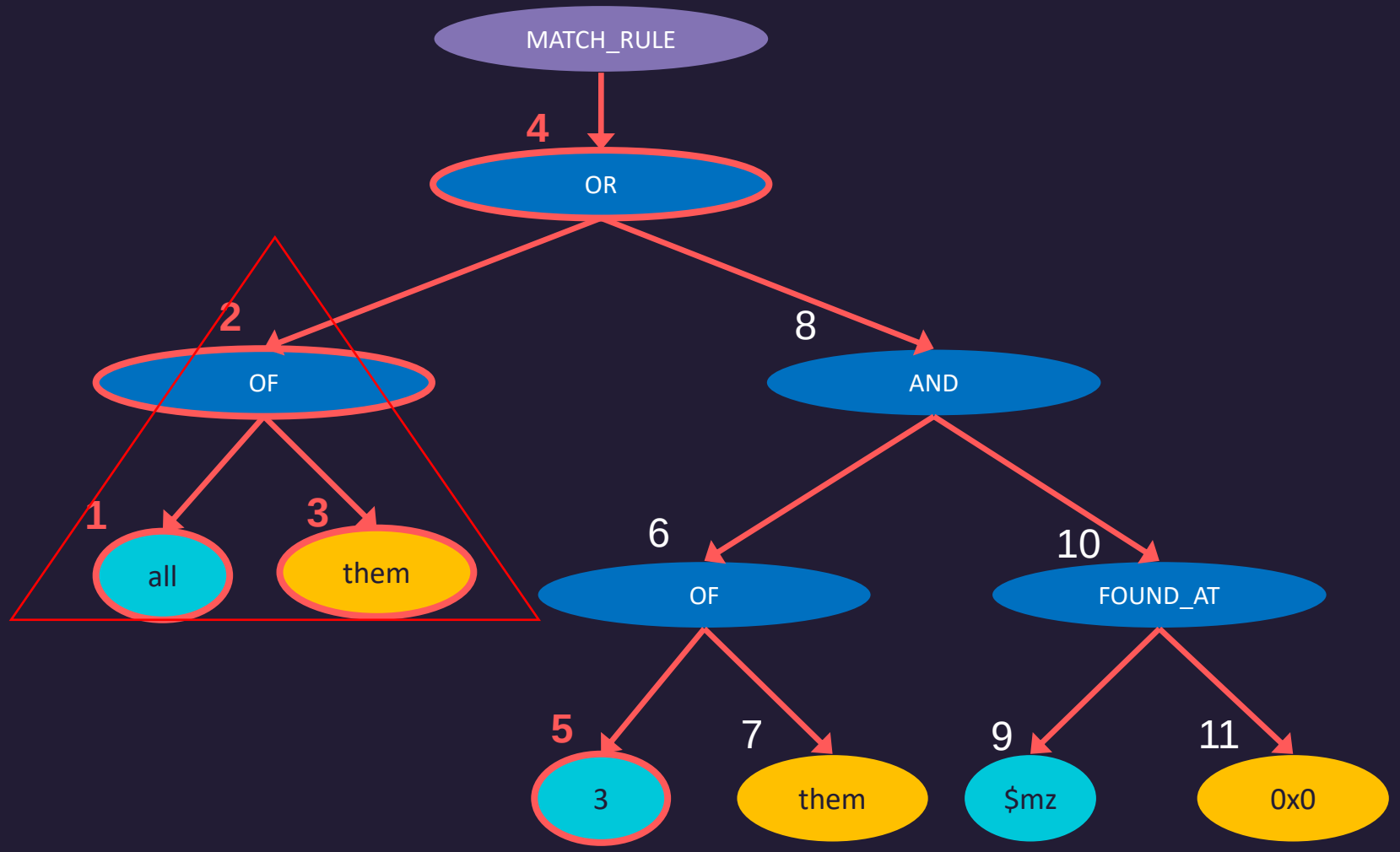
(all of them)

Output Rule



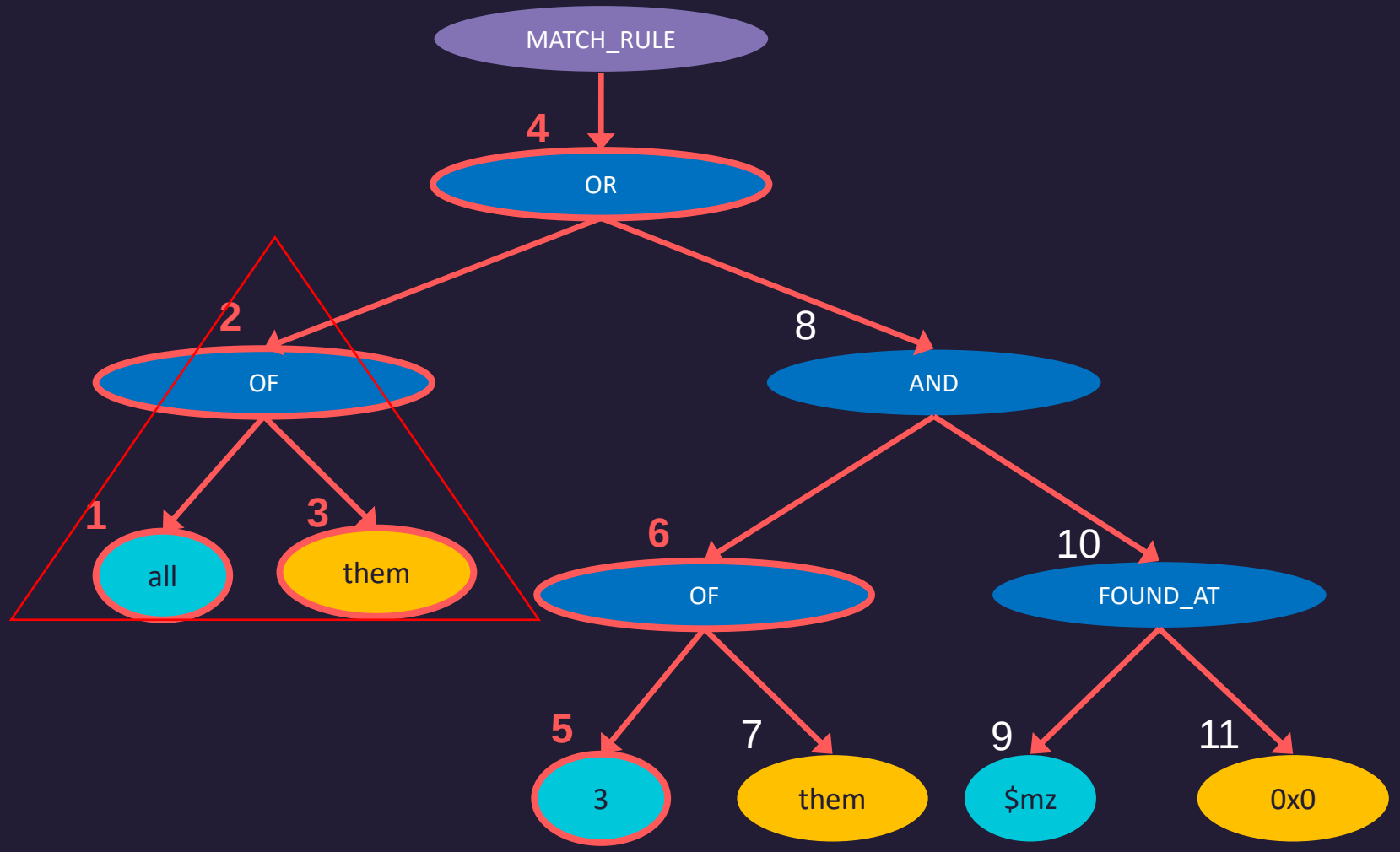
(all of them) **or** ()

Output Rule



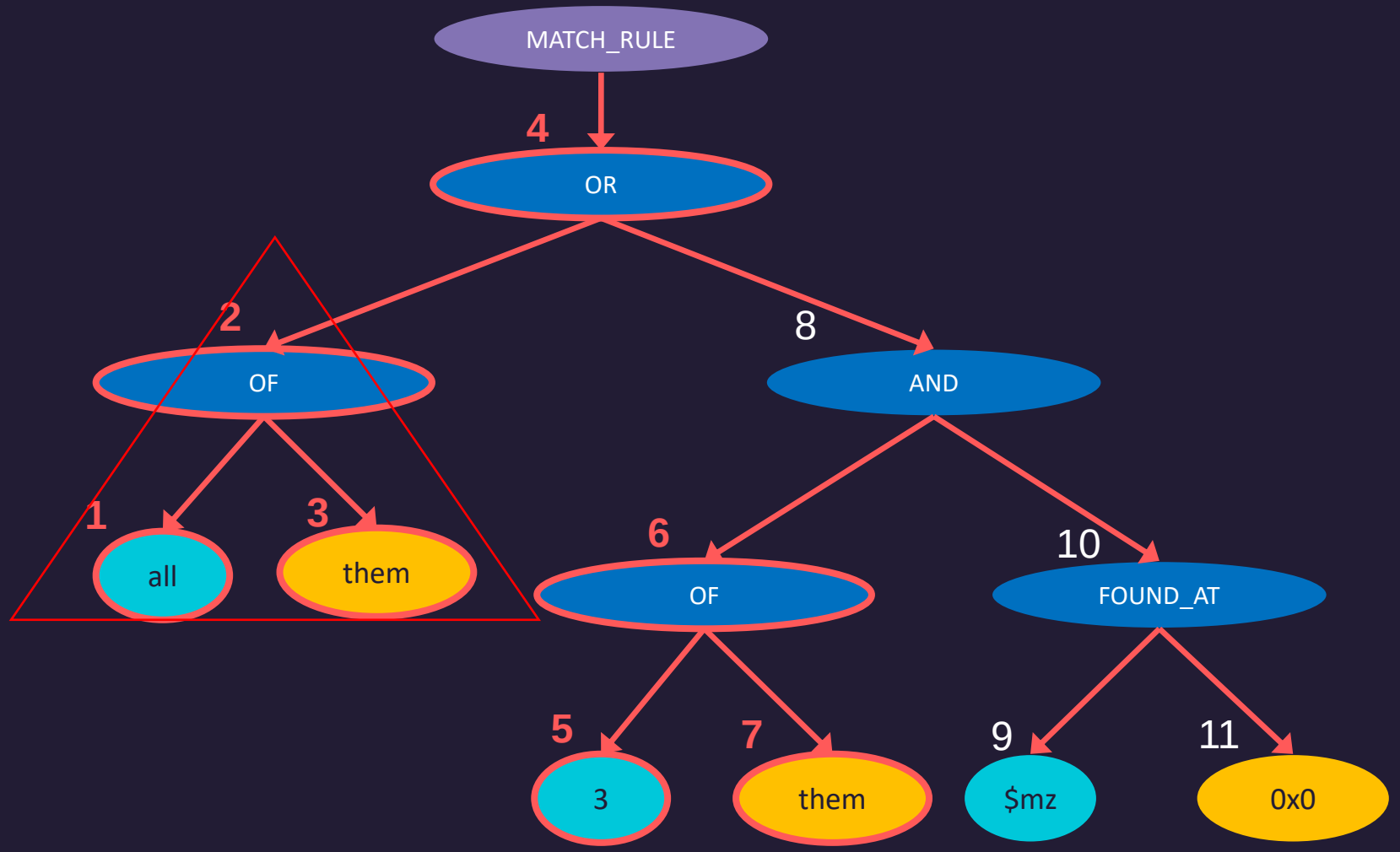
(all of them) or ((3))

Output Rule



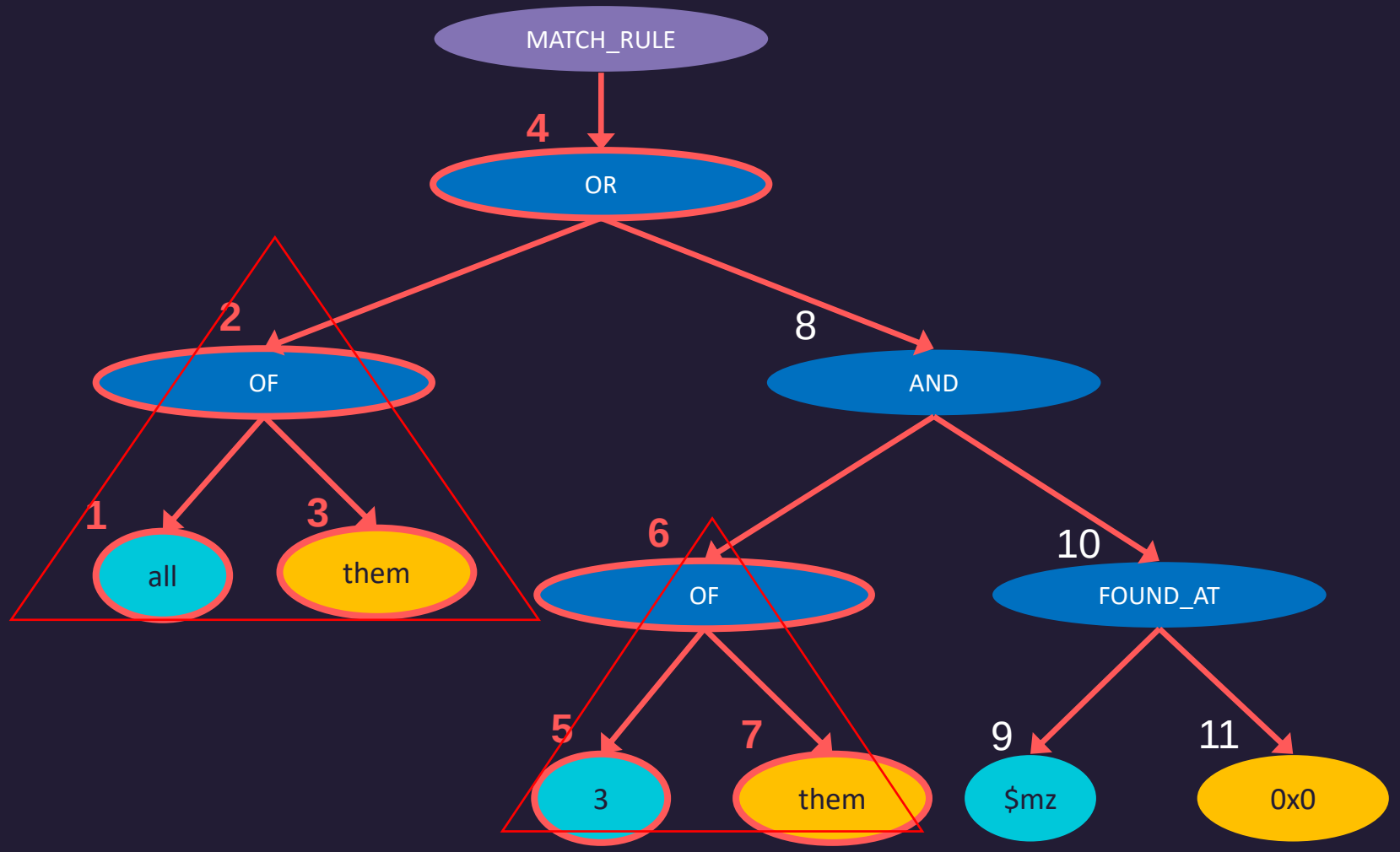
(all of them) or ((3 **of**))

Output Rule



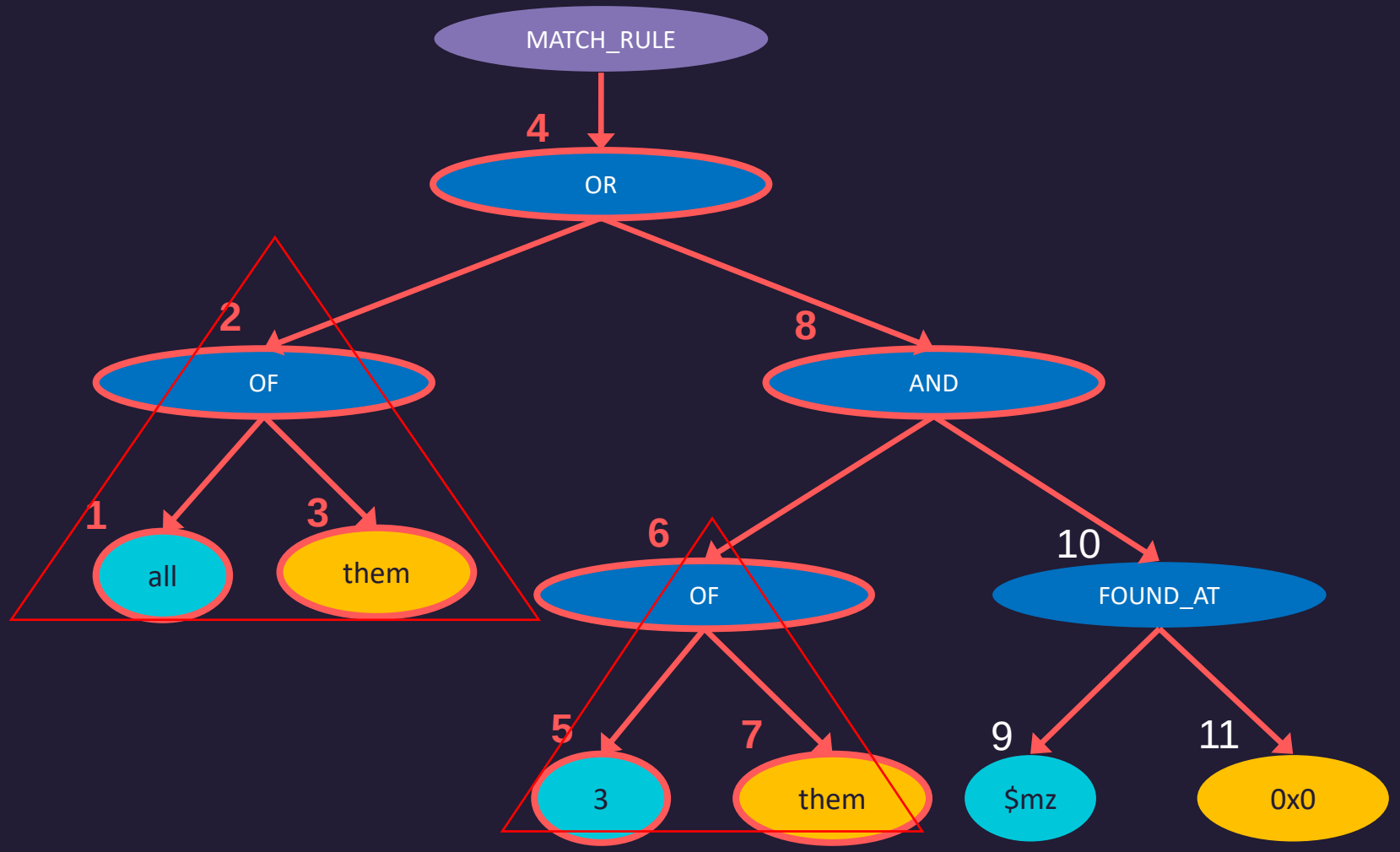
(all of them) or ((3 of **them**))

Output Rule



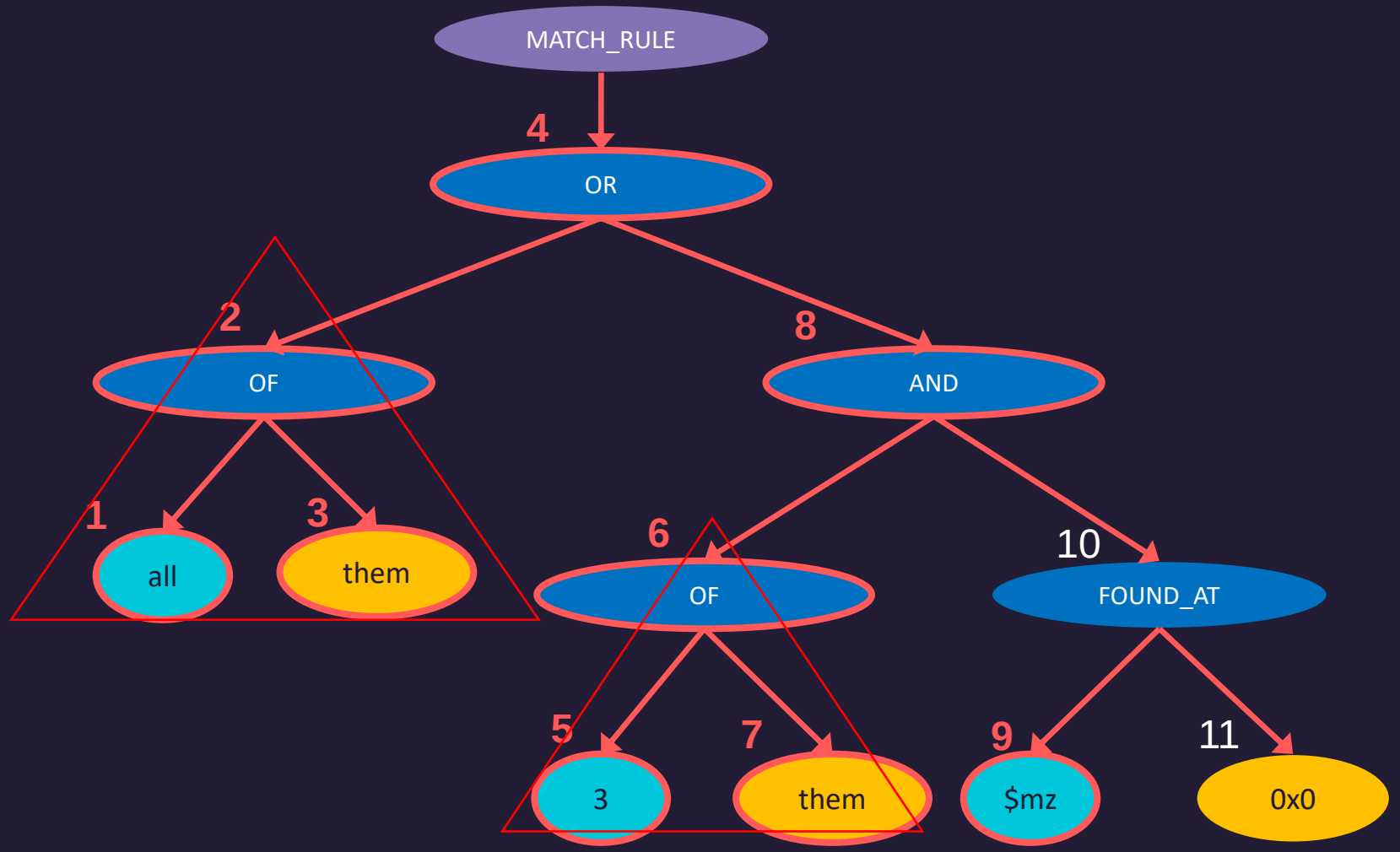
(all of them) or ((3 of them))

Output Rule



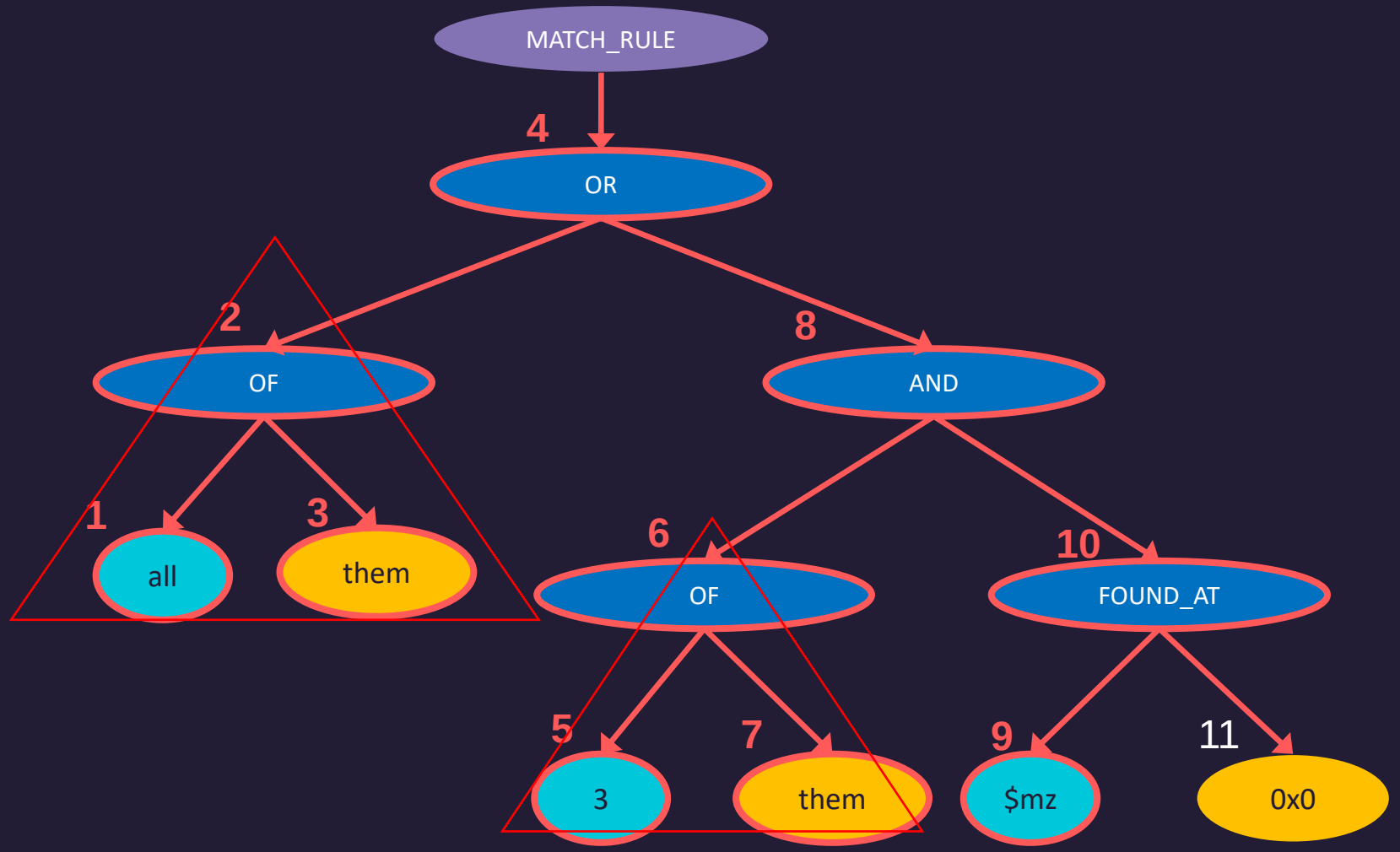
(all of them) or ((3 of them) **and** ())

Output Rule



(all of them) or ((3 of them) and ($mz))

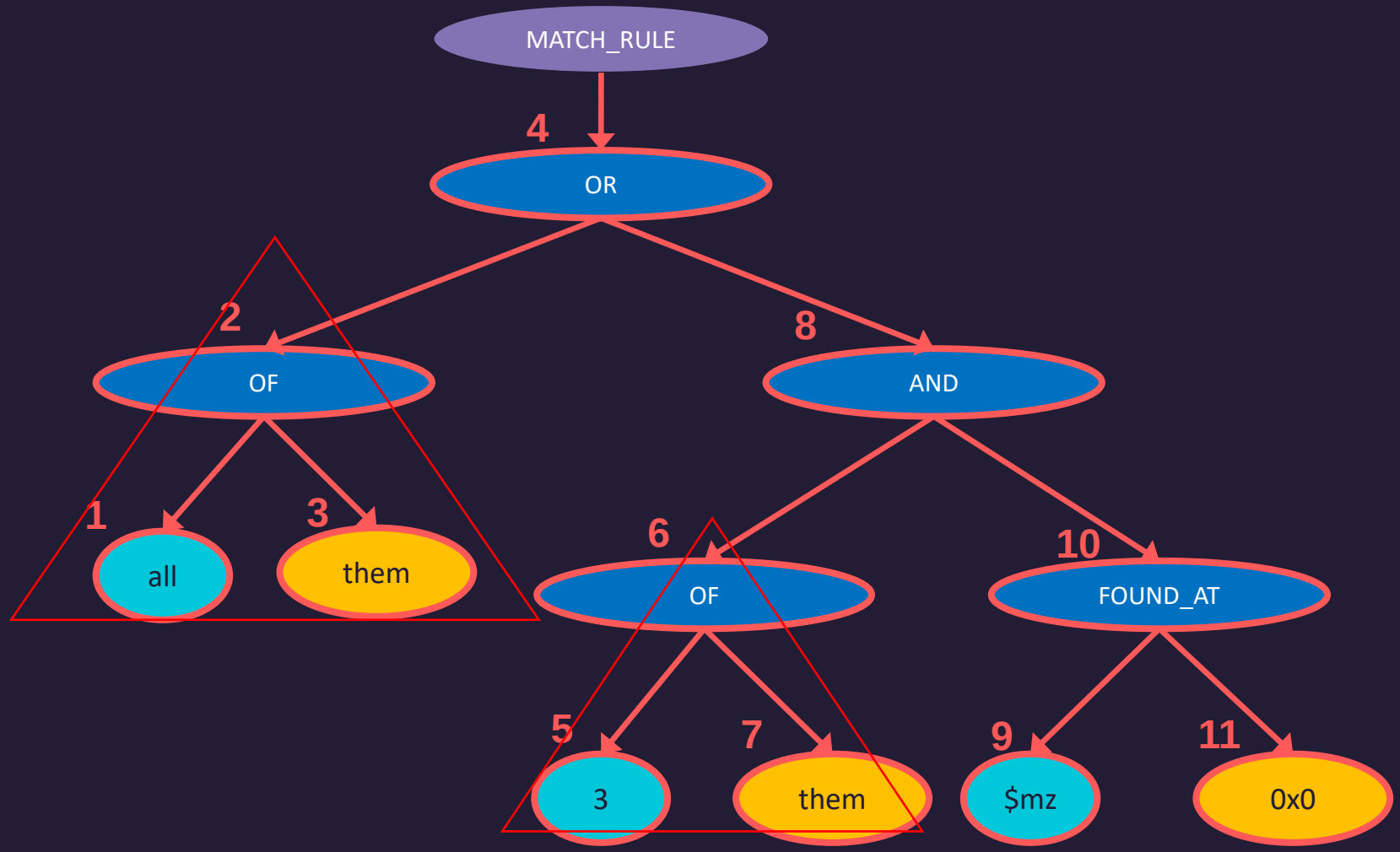
Output Rule



(all of them) or ((3 of them) and ($mz **at**))

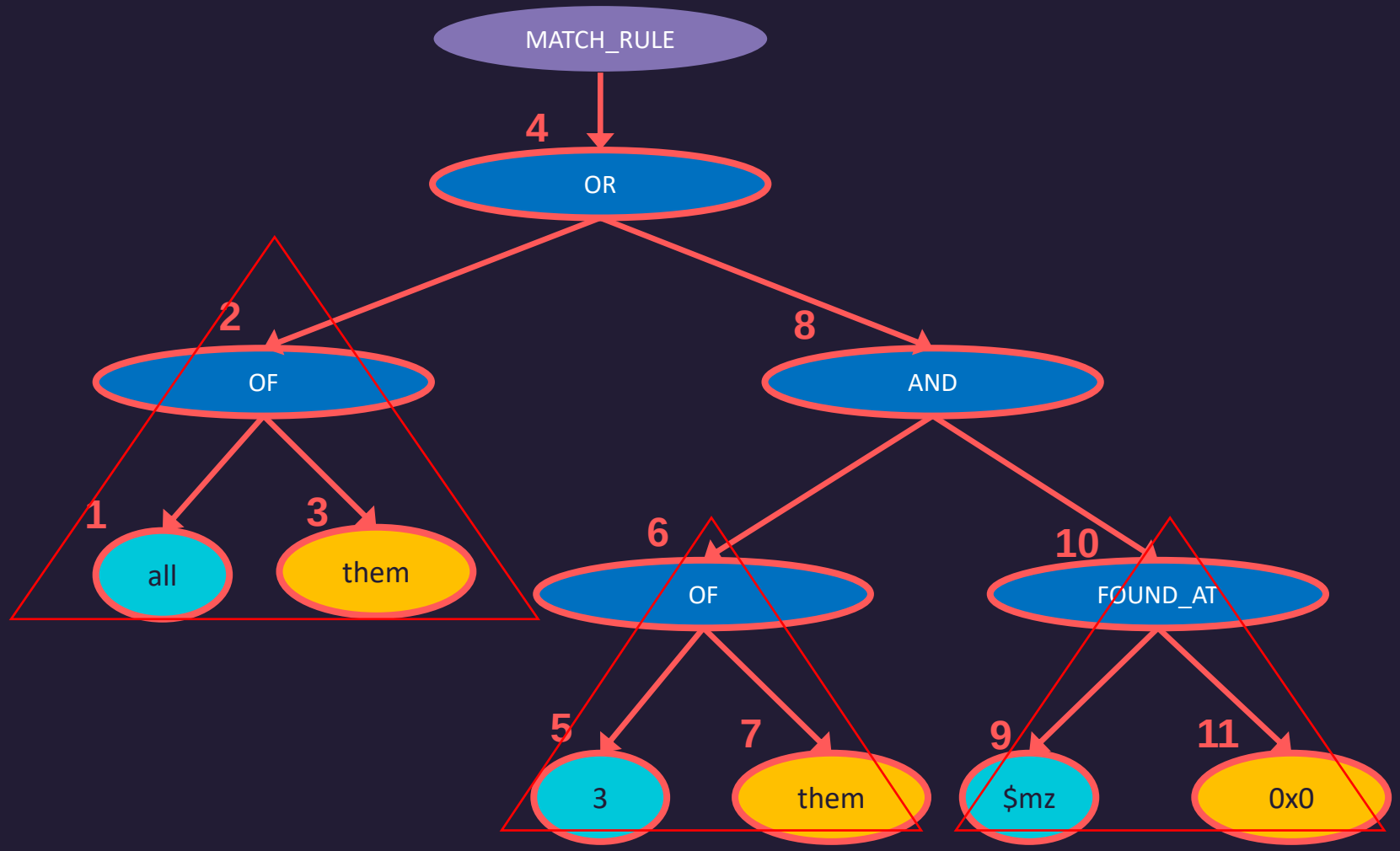
CYCRAFT

Output Rule



(all of them) or ((3 of them) and ($mz at **0**))

Output Rule



(all of them) or ((3 of them) and ($mz at 0))

Section Summary

- Yara uses a virtual machine to execute rules
- Decompilation is achieved by scanning through the bytecode and keep an argument stack to simulate the execution to generate the AST
- Inorder-traverse the AST to output rule

Pattern Decompilation

```
RE_OPCODE_LITERAL ( 0x6 )
RE_OPCODE_LITERAL ( 0x7 )
RE_OPCODE_LITERAL ( 0x8 )
RE_OPCODE_LITERAL ( 0x9 )
RE_OPCODE_LITERAL ( 0xa )
RE_OPCODE_REPEAT_ANY_UNGREEDY ( 0x1 0xa )
RE_OPCODE_LITERAL ( 0xb )
RE_OPCODE_LITERAL ( 0xc )
RE_OPCODE_MATCH
```

{06 07 08 09 0a [1-10] 0b 0c}

Regex Bytecode Basic Instruction Set

- Based on Regular Expression Matching: the Virtual Machine Approach
- No optimization pass

Instruction	Description
LITERAL x	Check the character pointed by the SP
JMP x	Jump to label x
SPLIT x, y	Split execution into 2 threads. One thread continues with x while the other with y
MATCH	Match is found, stop this thread

Generating Regex Bytecode

Pattern	Emitted Code
$e_1 e_2$	{codes for e1} {codes for e2}
$e_1 \mid e_2$	split L1, L2 L1: {codes for e1} jmp L3 L2: {codes for e2} L3:
$e?$	split L1, L2 L1: {codes for e} L2:
$e*$	L1: split L2, L3 L2: {codes for e} jmp L1 L3:
$e+$	L1: {codes for e} split L1, L3 L3:

Matching

Pattern

a+b+

Input String

aab

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Threads

T1 aab

Matching

Pattern

a+b+

Input String

aab

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Threads

T1 aab Match

Matching

Pattern

a+b+

Input String

aab

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Threads

T1 aab Split

Matching

Pattern

a+b+

Input String

aab

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Threads

T1	aab	Match
T2	aab	No Match

Matching

Pattern

a+b+

Input String

aab

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Threads

T1	aab	Split
T2	aab	Died

Matching

Pattern

a+b+

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Input String

aab

Threads

T1	aa**b**	No Match
T2	aab	Died
T3	aa**b**	Match

Matching

Pattern

a+b+

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Input String

aab

Threads

T1	aab	Died
T2	aab	Died
T3	aab_	Split

Matching

Pattern

a+b+

Generated Bytecode

```
L1: LITERAL a
    SPLIT   L1, L2
L2: LITERAL b
    SPLIT   L2, L3
L3: MATCH
```

Input String

aab

Threads

T1	aab	Died
T2	aab	Died
T3	aab_	No Match
T4	aab_	Accept

Decompiling Patterns

- Extract matches from the AC automata
- Decompile forward and backward regex bytecode

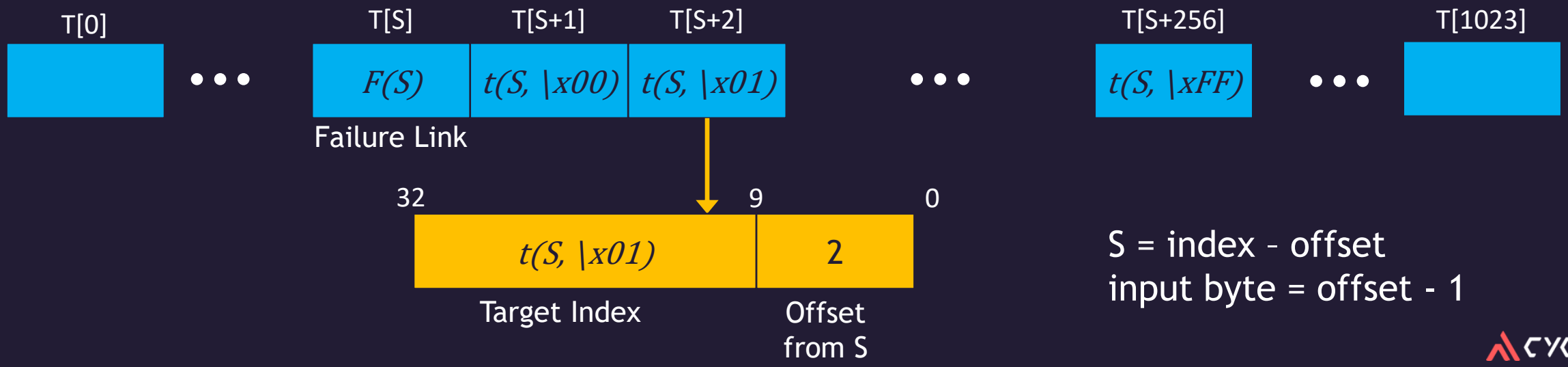
AC Automata Structure (v3.9.0)

In v3.4.0, Yara used a graph structure to stored the AC automata, while in v3.9.0, the automata is represented by a transition table and a match table

- Match Table: M[S] stores the linked list of matches of state S
- Transition Table: Implemented ACISM(Aho-Corasick Interleaved State-transition Matrix)
  - Transition table is NOT NEEDED for decompilation

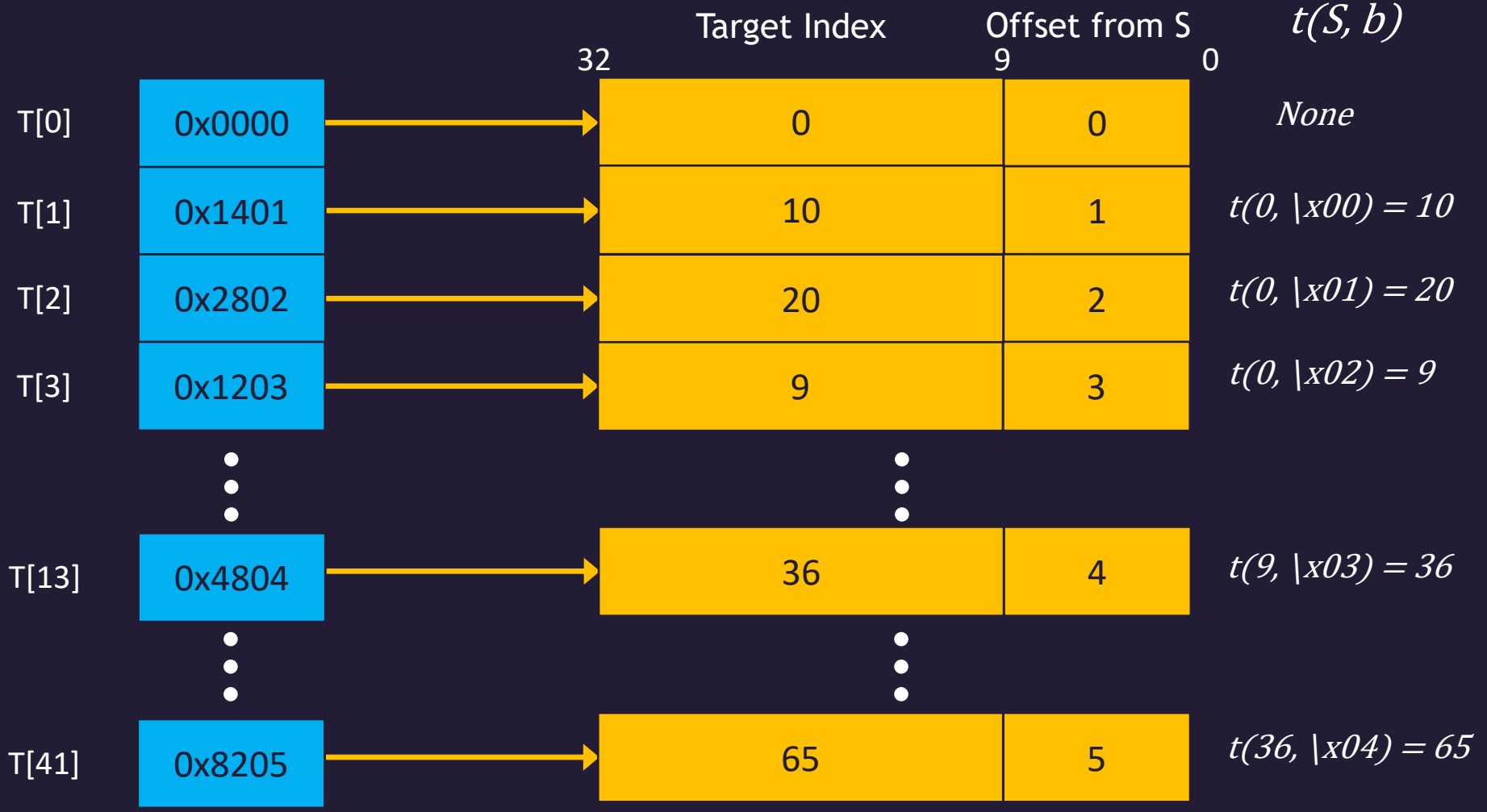
AC Automata Structure (v3.9.0)

- ACISM(Aho-Corasick Interleaved State-transition Matrix)
- Slot T[S+1+b] stores the information for transition $t(S, b)$, where S is the state and b is the input byte



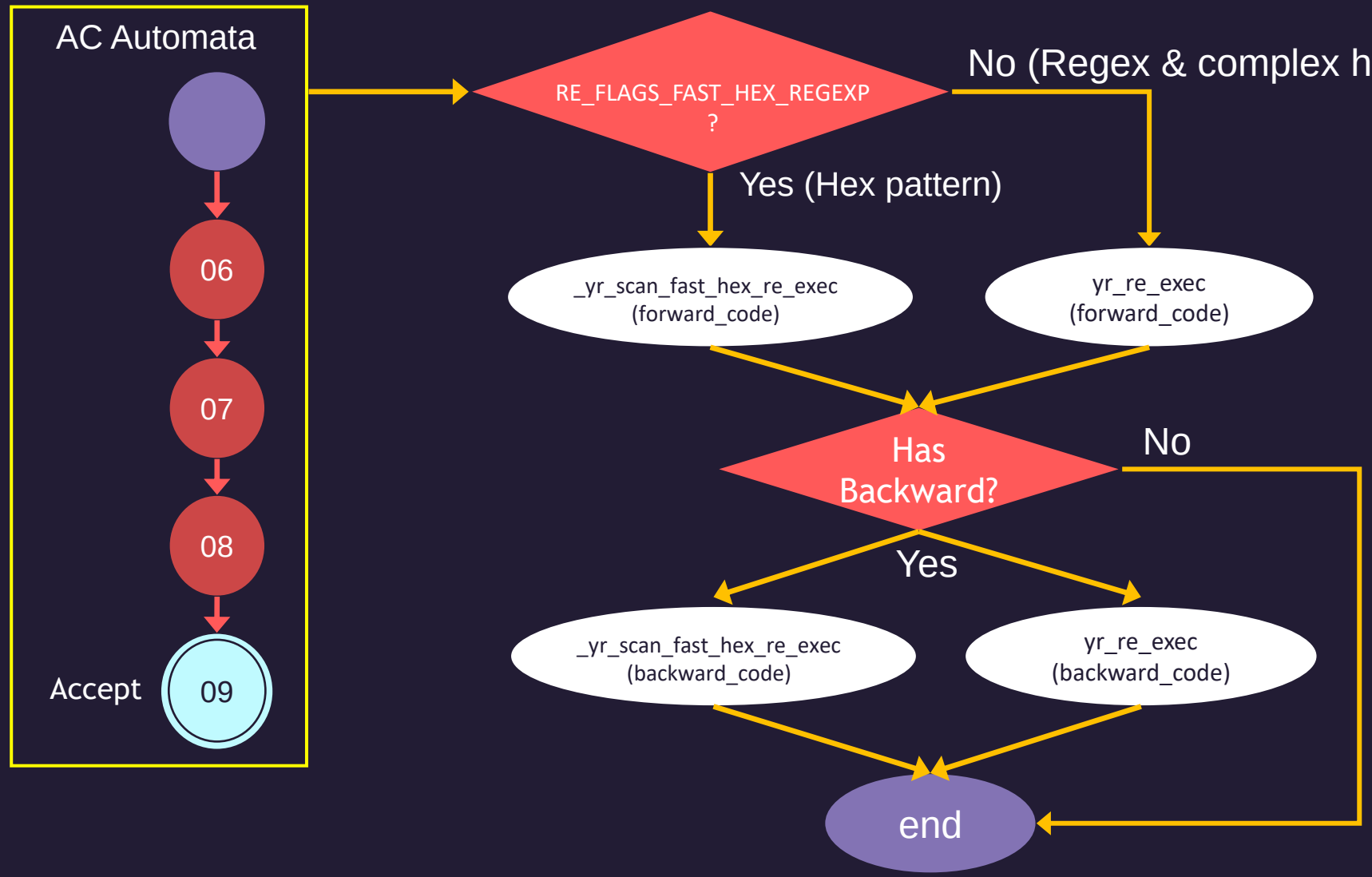
AC Automata Structure (v3.9.0)

S = index - offset
input byte = offset - 1

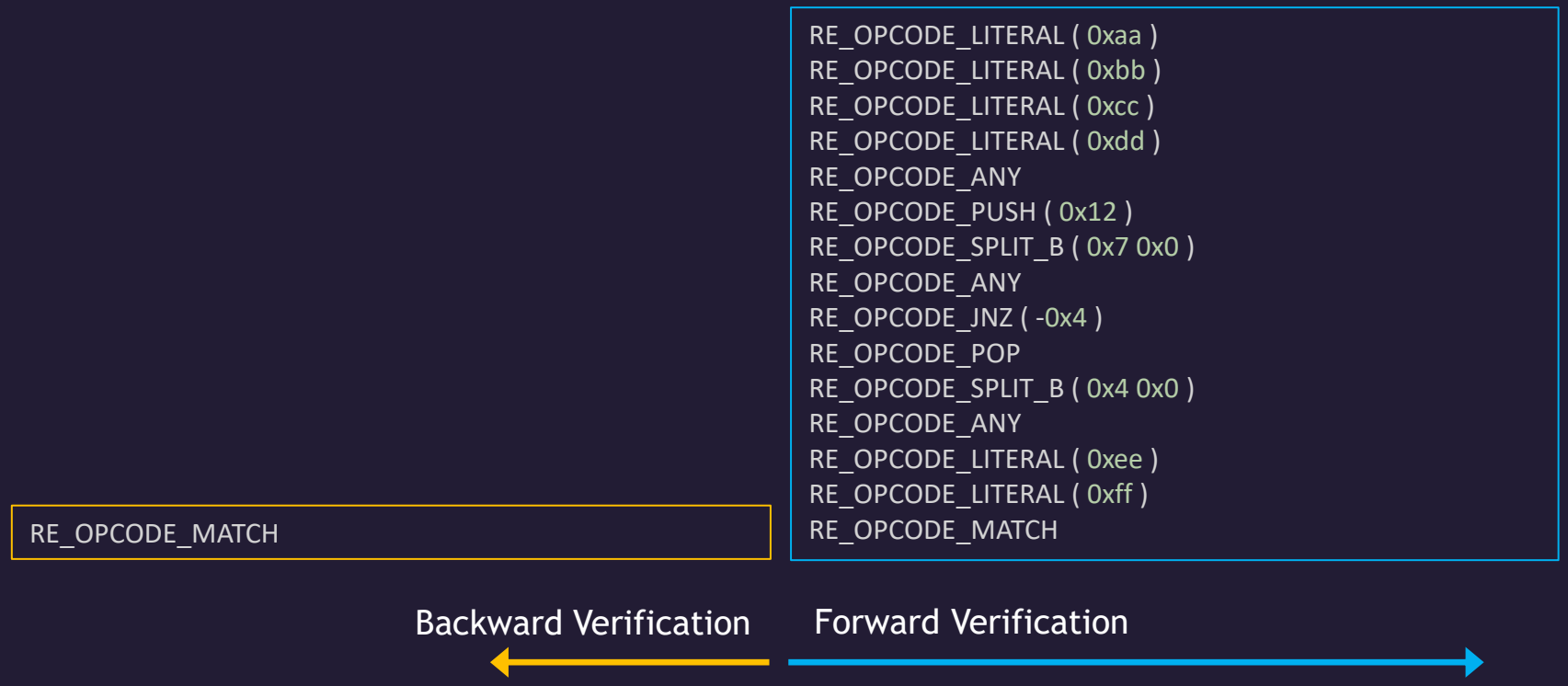


Scanning Workflow

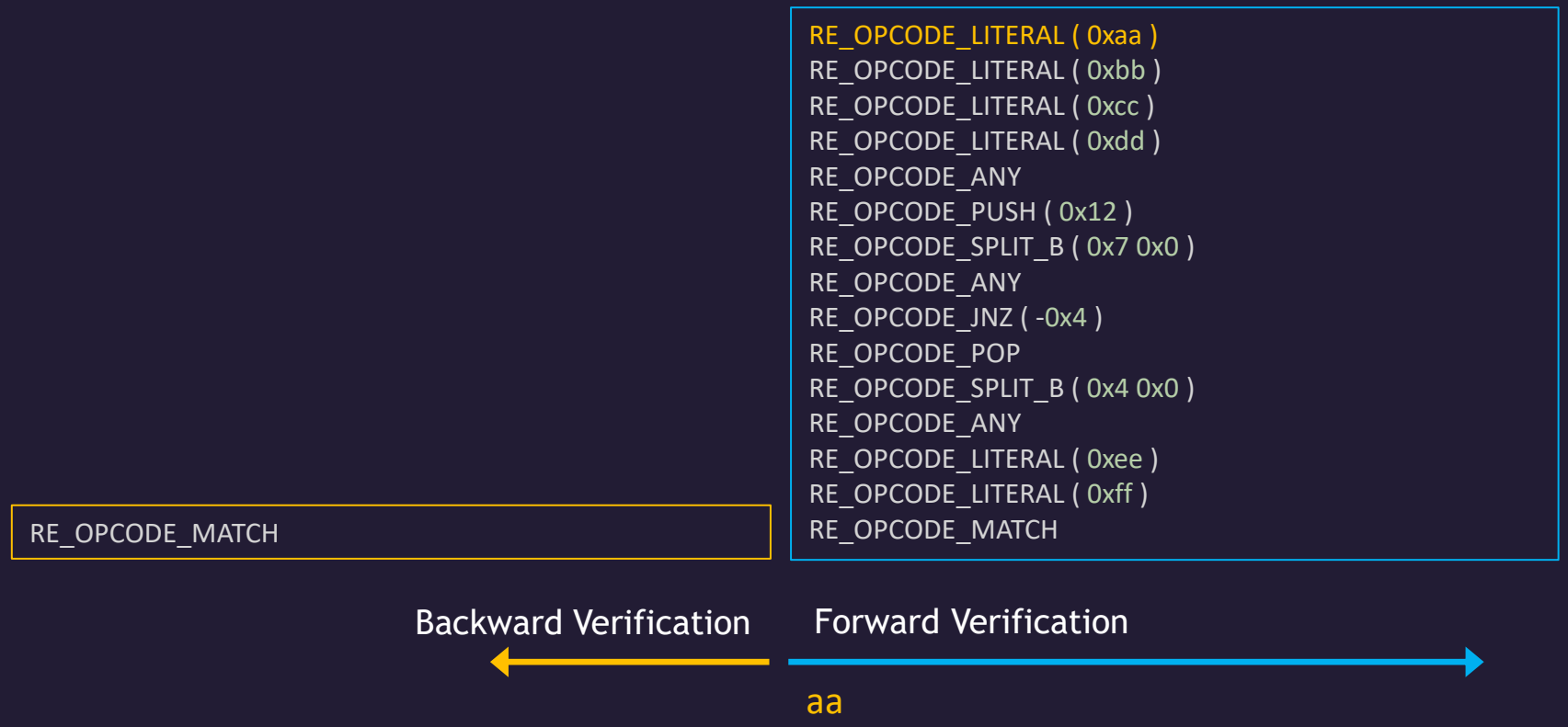
Hex Pattern: {00 1? [2-4] 00 0A}
Regular Exp: /asd.+*f{1,3}z?/



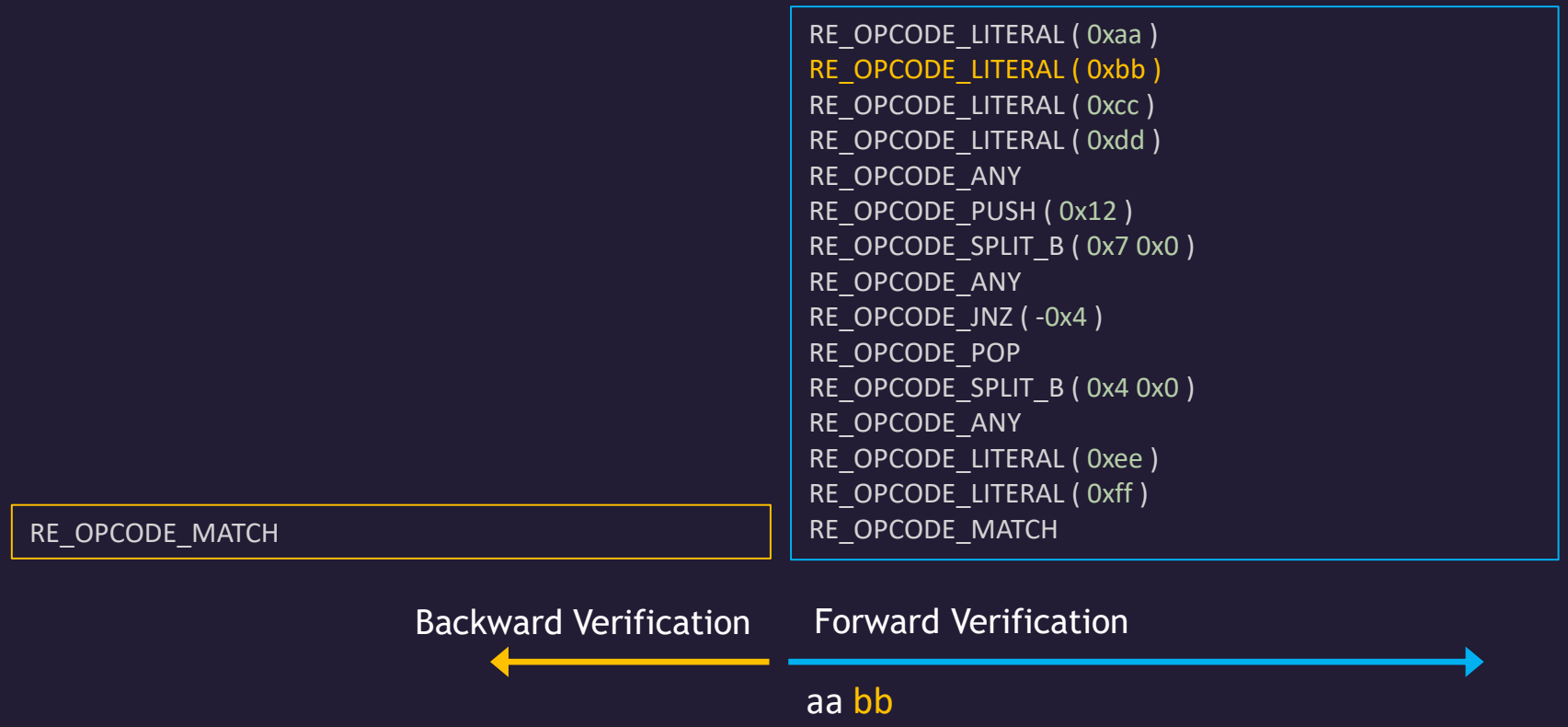
Decompiling Hex Patterns (v3.4.0)



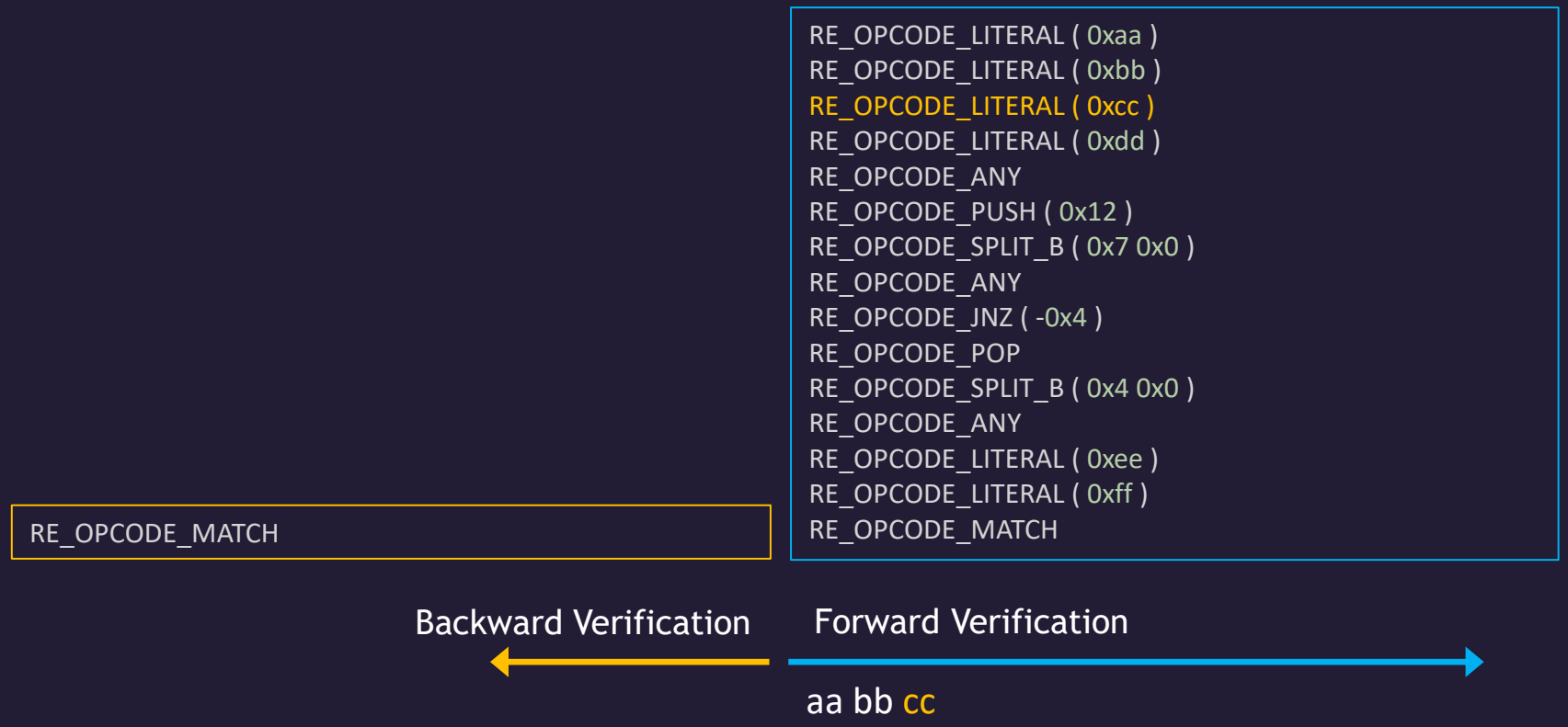
Decompiling Hex Patterns (v3.4.0)



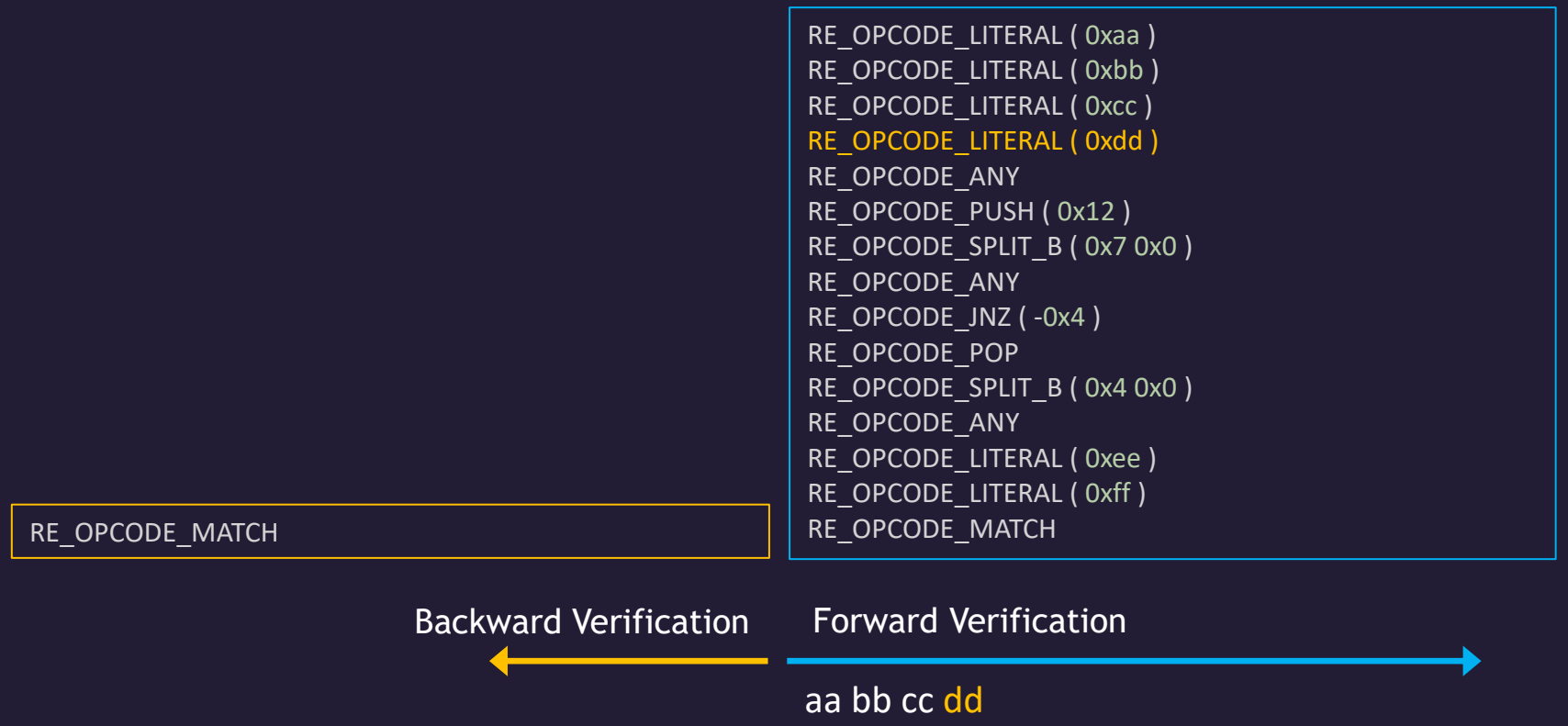
Decompiling Hex Patterns (v3.4.0)



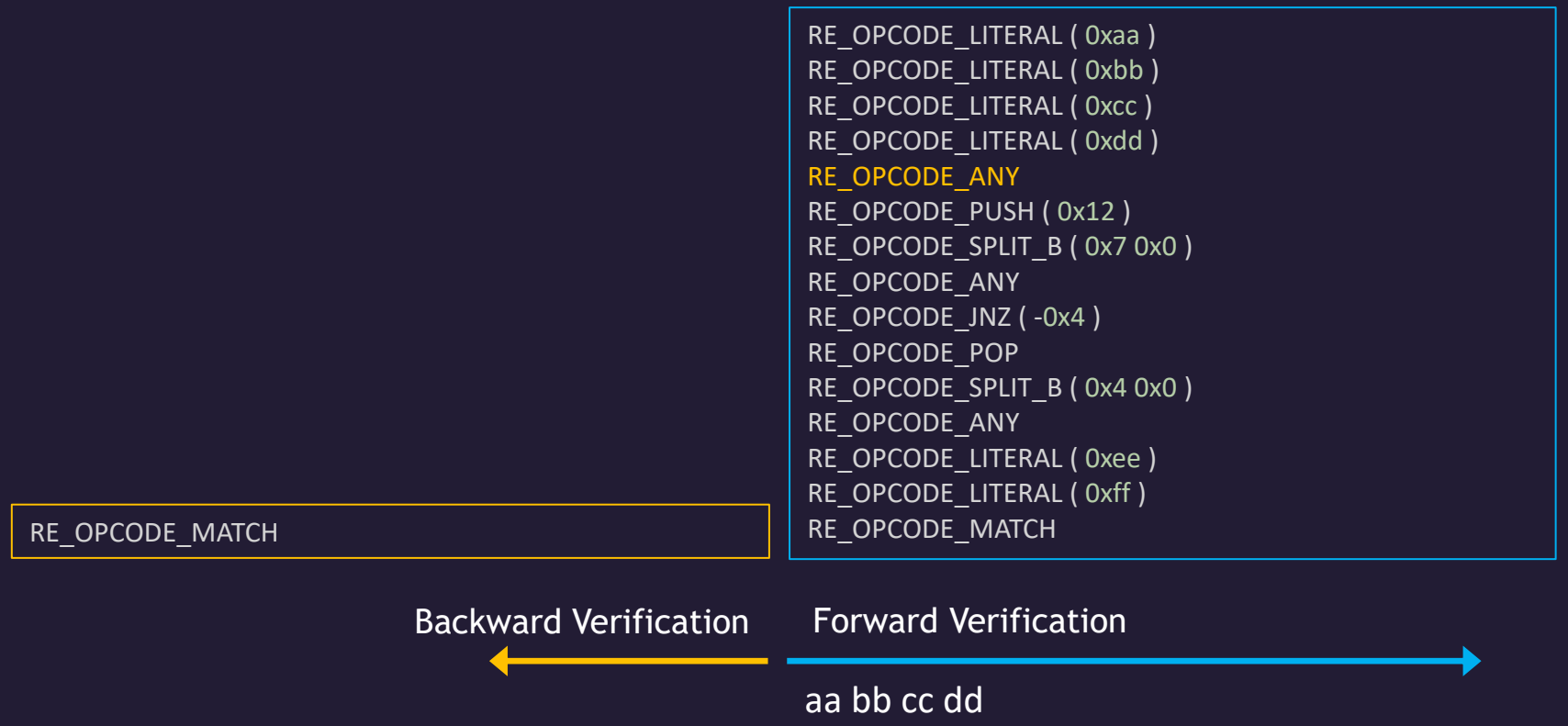
Decompiling Hex Patterns (v3.4.0)



Decompiling Hex Patterns (v3.4.0)



Decompiling Hex Patterns (v3.4.0)



Decompiling Hex Patterns (v3.4.0)

Code for e{n,m} looks like:

```
    code for e      (repeated n times)
    push m-n-1      (3 bytes long)
L0: split L1, L2    (3 bytes long)
L1: any             (1 byte long)
    jnz L0          (3 bytes long)
L2: pop             (1 byte long)
    split L3, L4
L3: code for e
L4:
```

RE_OPCODE_MATCH

m

```
RE_OPCODE_LITERAL ( 0xaa )
RE_OPCODE_LITERAL ( 0xbb )
RE_OPCODE_LITERAL ( 0xcc )
RE_OPCODE_LITERAL ( 0xdd )
RE_OPCODE_ANY                      } n = 1
RE_OPCODE_PUSH ( 0x12 )
RE_OPCODE_SPLIT_B ( 0x7 0x0 )      } m-n-1
RE_OPCODE_ANY                      }
RE_OPCODE_JNZ ( -0x4 )             }
RE_OPCODE_POP
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY                      } 1
RE_OPCODE_LITERAL ( 0xee )
RE_OPCODE_LITERAL ( 0xff )
RE_OPCODE_MATCH
```

m-n-1 = 0x12 = 18

m = (m-n-1) + n + 1

= 18 + 1 + 1

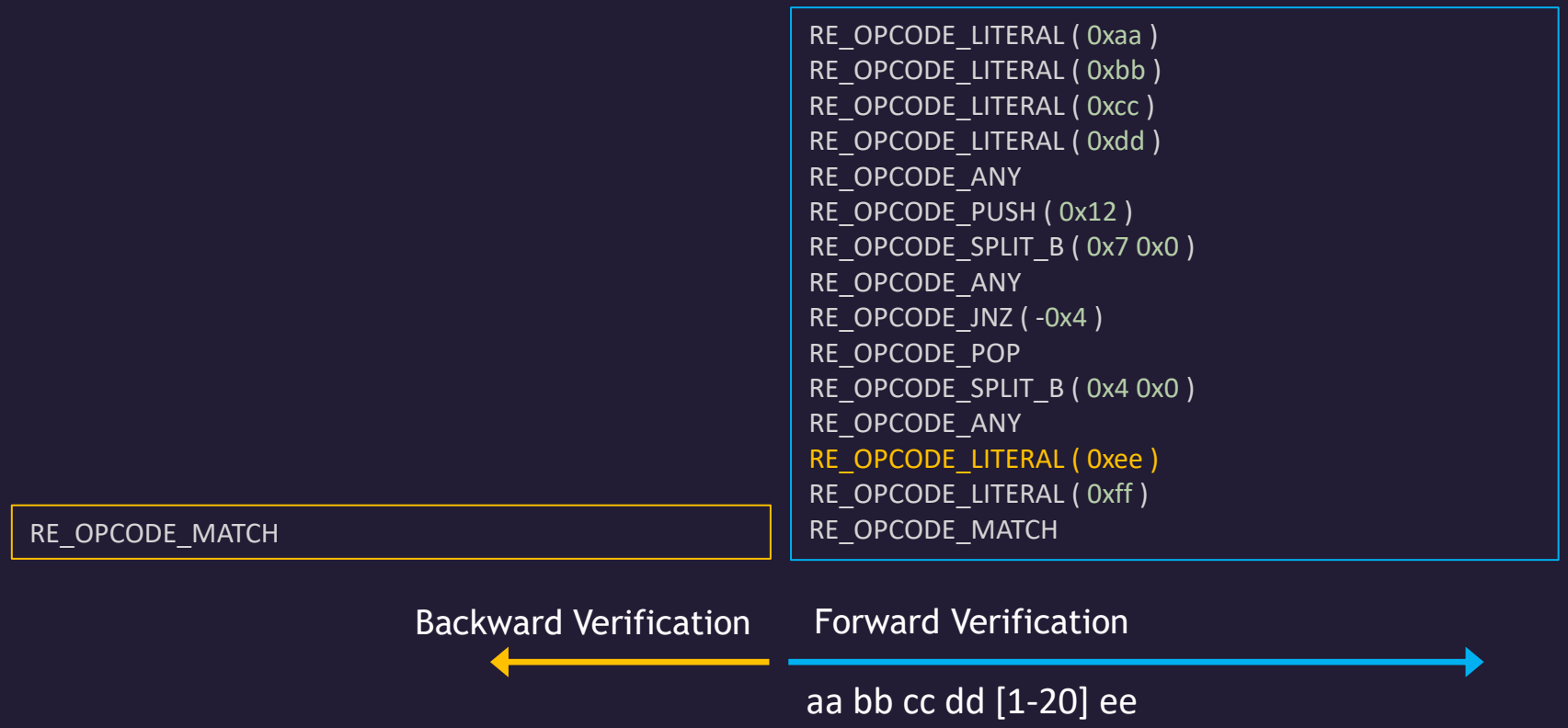
= 20

Backward Verification ← → Forward Verification

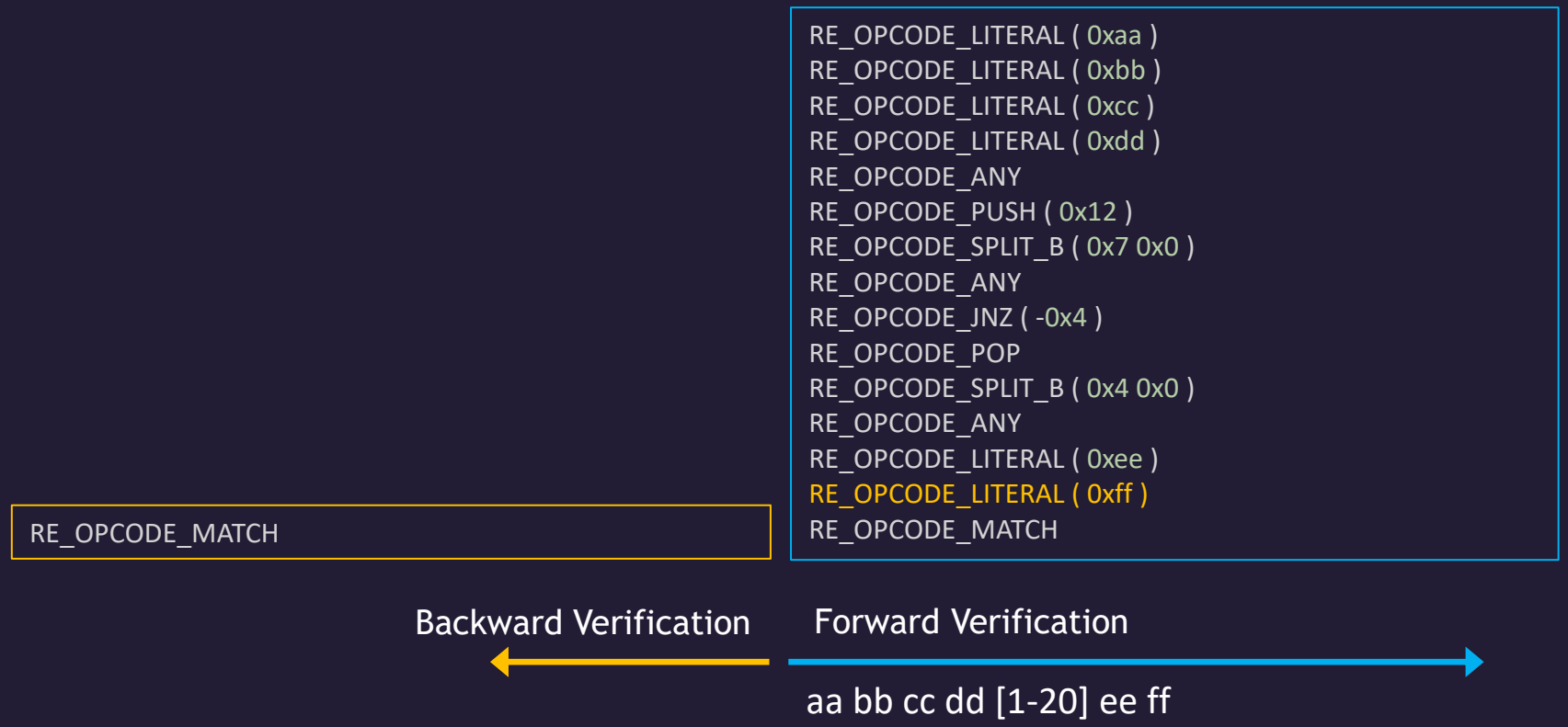
aa bb cc dd [1-20]

CYCRAFT

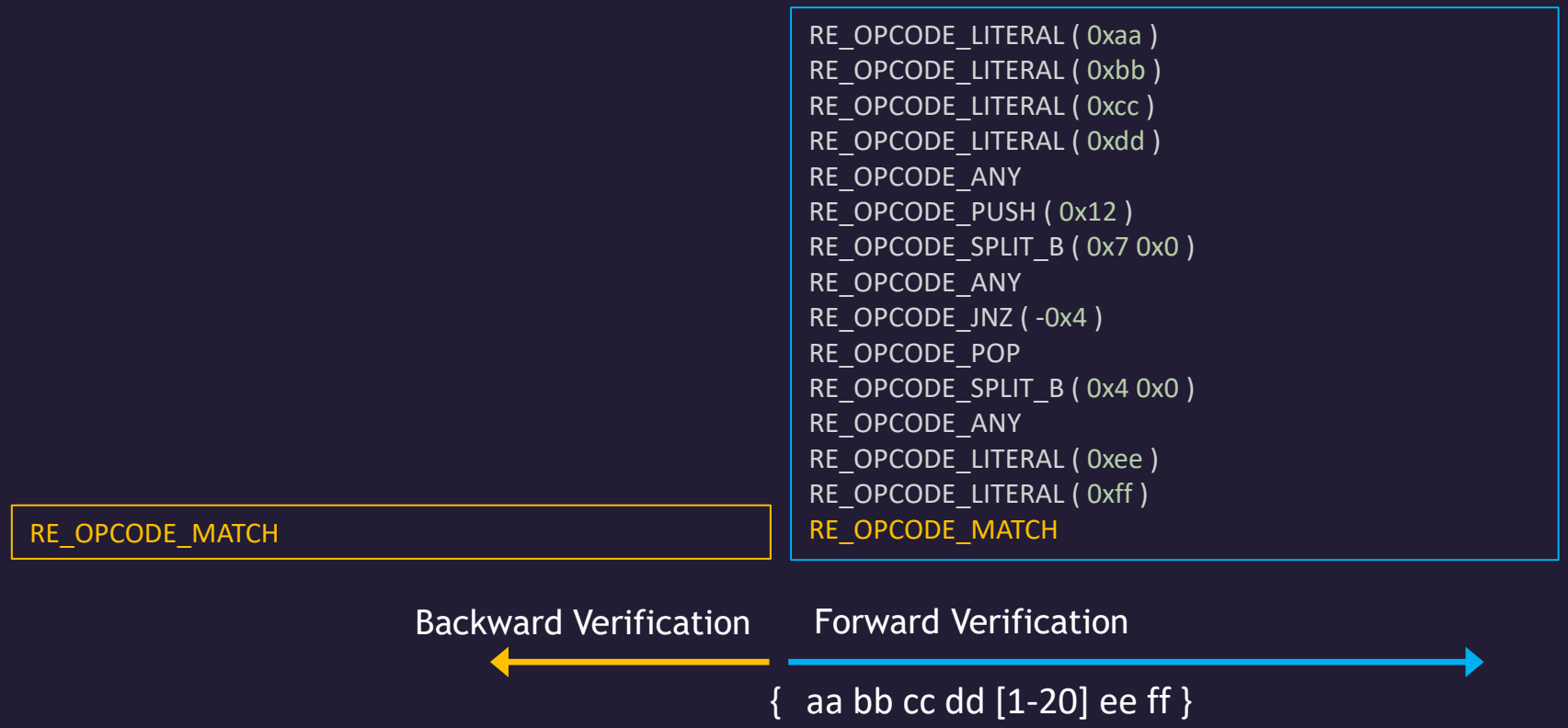
Decompiling Hex Patterns (v3.4.0)



Decompiling Hex Patterns (v3.4.0)



Decompiling Hex Patterns (v3.4.0)



Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

78

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

78

Decompiling Hex Patterns (v3.4.0)

10 times

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

SPLIT_B occurs without PUSH only when range = [N, N+1]
The code would be like:

```
any      (repeated n times)
split_b
any
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

[10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

2nd arg: 0xf0 => X?
2nd arg: 0x0f => ?X

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

n (+0xc)

m (+0x6)

SPLIT_A occurs if there is (A | B) structure

```
code:
    split_a L1, 0
    code for A      ----- n
    jmp L2
L1:  code for B     ----- m
L2:  ...
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

(|) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

n (+0xc)

m (+0x6)

SPLIT_A occurs if there is (A | B) structure

```
code:
    split_a L1, 0
    code for A      ----- n
    jmp L2
L1:  code for B     ----- m
L2:  ...
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

(30 |) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

n (+0xc)

m (+0x6)

SPLIT_A occurs if there is (A | B) structure

```
code:
    split_a L1, 0
    code for A      ----- n
    jmp L2
L1:  code for B     ----- m
L2:  ...
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

(20 30 |) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

n (+0xc)

m (+0x6)

SPLIT_A occurs if there is (A | B) structure

```
code:
    split_a L1, 0
    code for A      ----- n
    jmp L2
L1:  code for B     ----- m
L2:  ...
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

(10 20 30 |) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

n (+0xc)

m (+0x6)

SPLIT_A occurs if there is (A | B) structure

```
code:
    split_a L1, 0
    code for A      ----- n
    jmp L2
L1:  code for B     ----- m
L2:  ...
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

(10 20 30 | 3?) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

2? (10 20 30 | 3?) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

?3 2? (10 20 30 | 3?) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

Backward Verification

Forward Verification

00 ?3 2? (10 20 30 | 3?) 1? [10-11] 78

CYCRAFT

Decompiling Hex Patterns (v3.4.0)

```
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_ANY
RE_OPCODE_SPLIT_B ( 0x4 0x0 )
RE_OPCODE_ANY
RE_OPCODE_MASKED_LITERAL ( 0x10 0xf0 )
RE_OPCODE_SPLIT_A ( 0xc 0x0 )
RE_OPCODE_LITERAL ( 0x30 )
RE_OPCODE_LITERAL ( 0x20 )
RE_OPCODE_LITERAL ( 0x10 )
RE_OPCODE_JUMP ( 0x6 )
RE_OPCODE_MASKED_LITERAL ( 0x30 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x20 0xf0 )
RE_OPCODE_MASKED_LITERAL ( 0x3 0xf )
RE_OPCODE_LITERAL ( 0x0 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_LITERAL 0x78
RE_OPCODE_MATCH
```

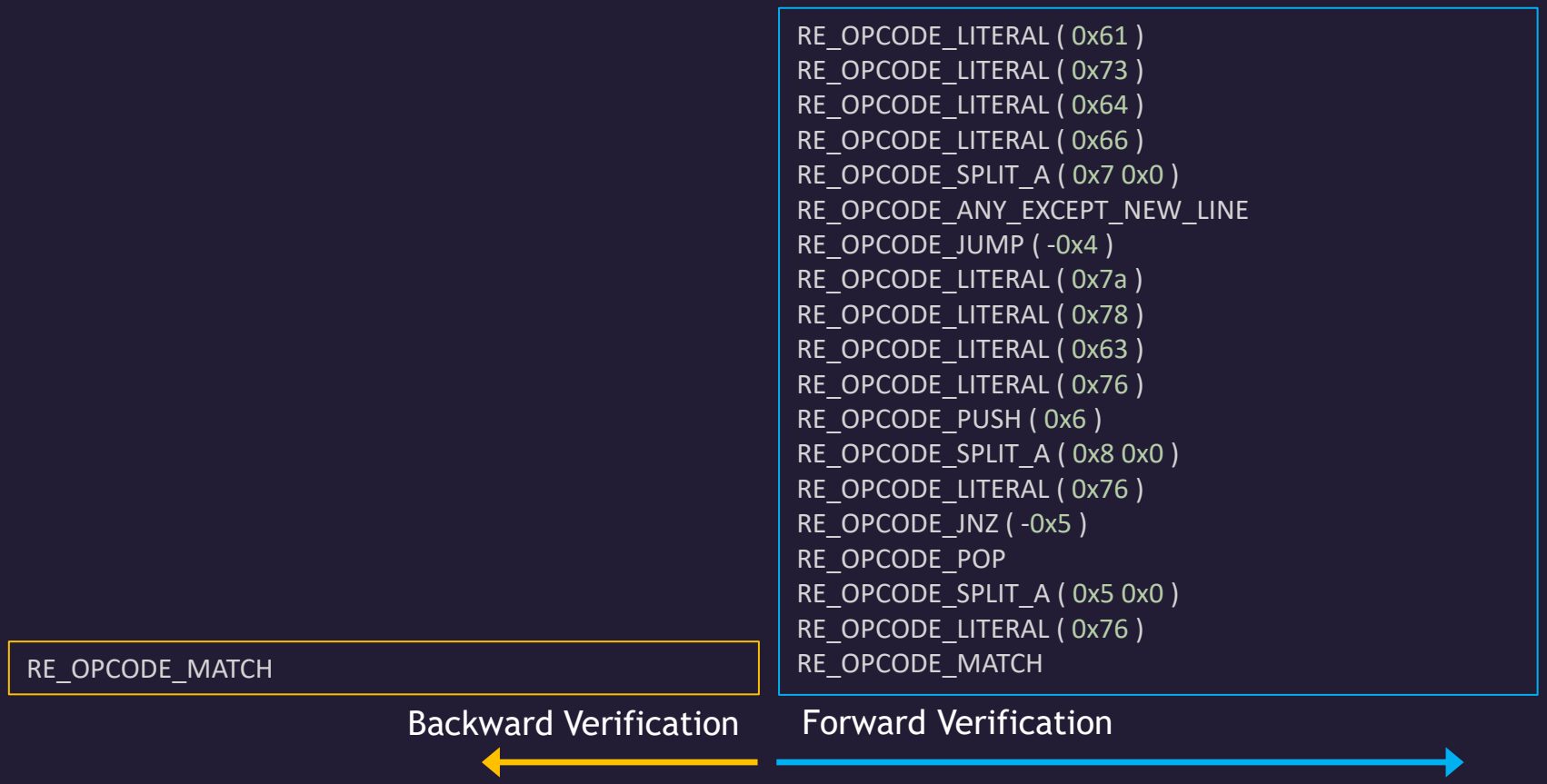
Backward Verification

Forward Verification

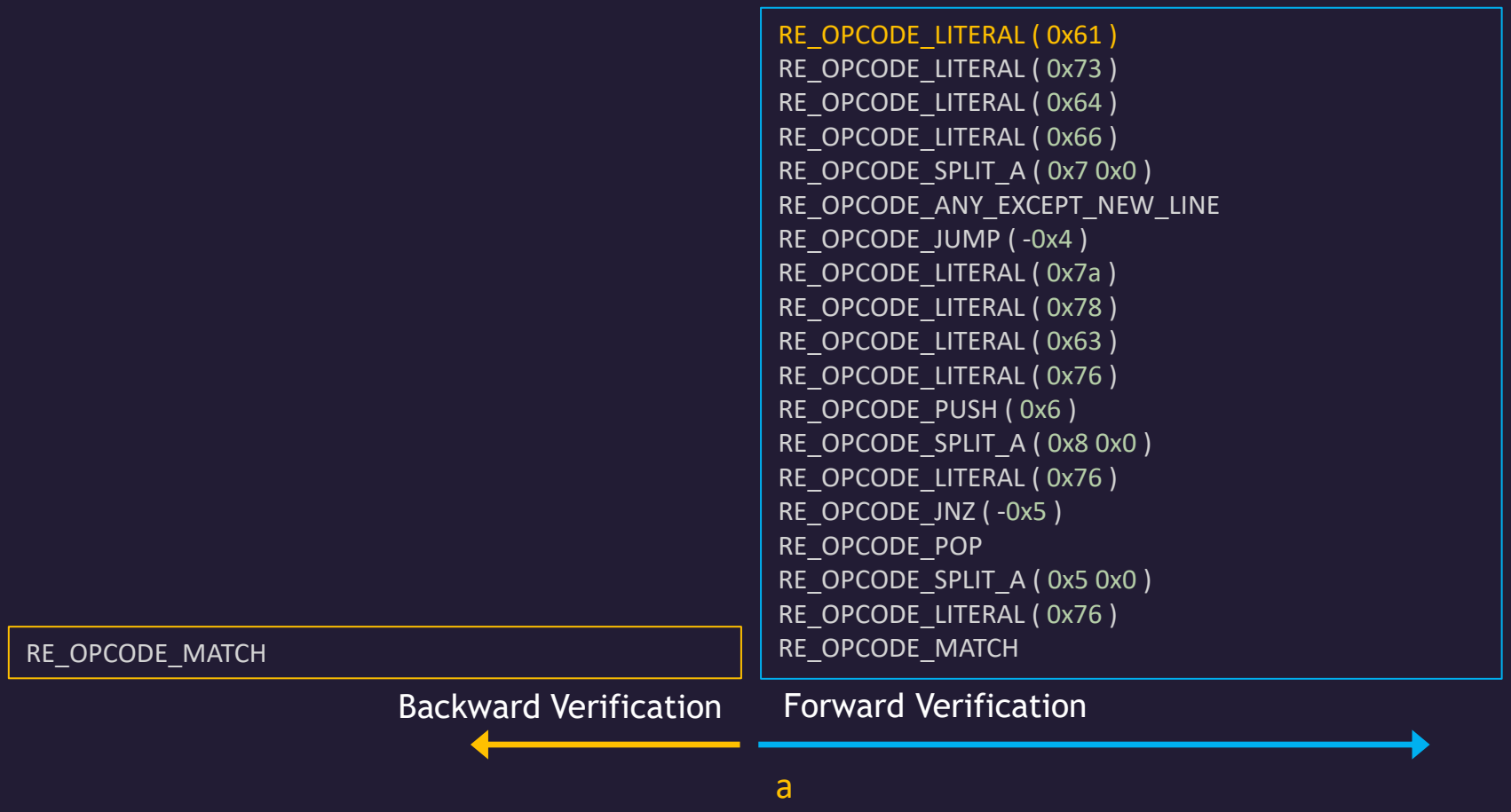
{ 00 ?3 2? (10 20 30 | 3?) 1? [10-11] 78 }

CYCRAFT

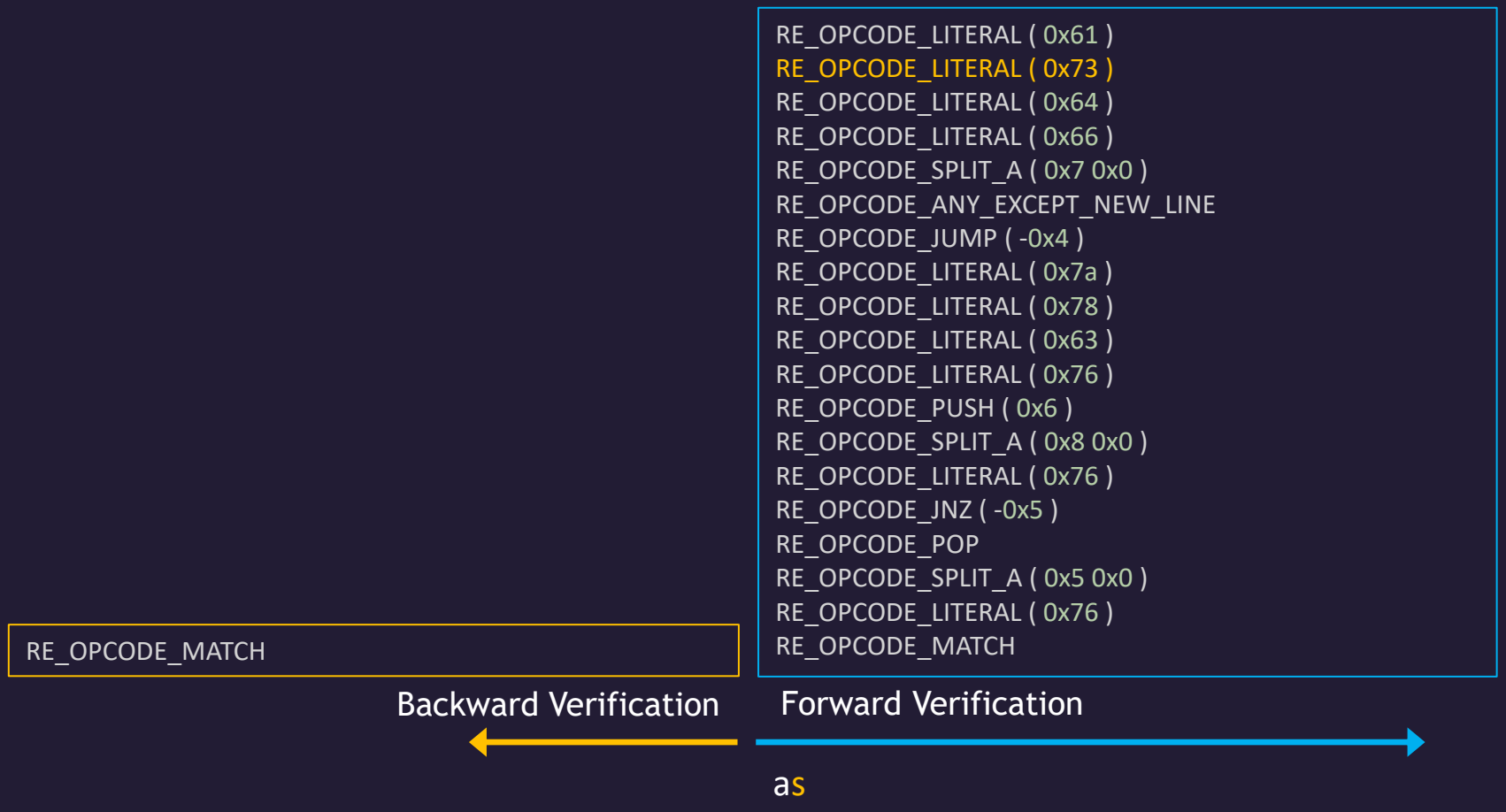
Decompiling Regular Expressions (v3.4.0)



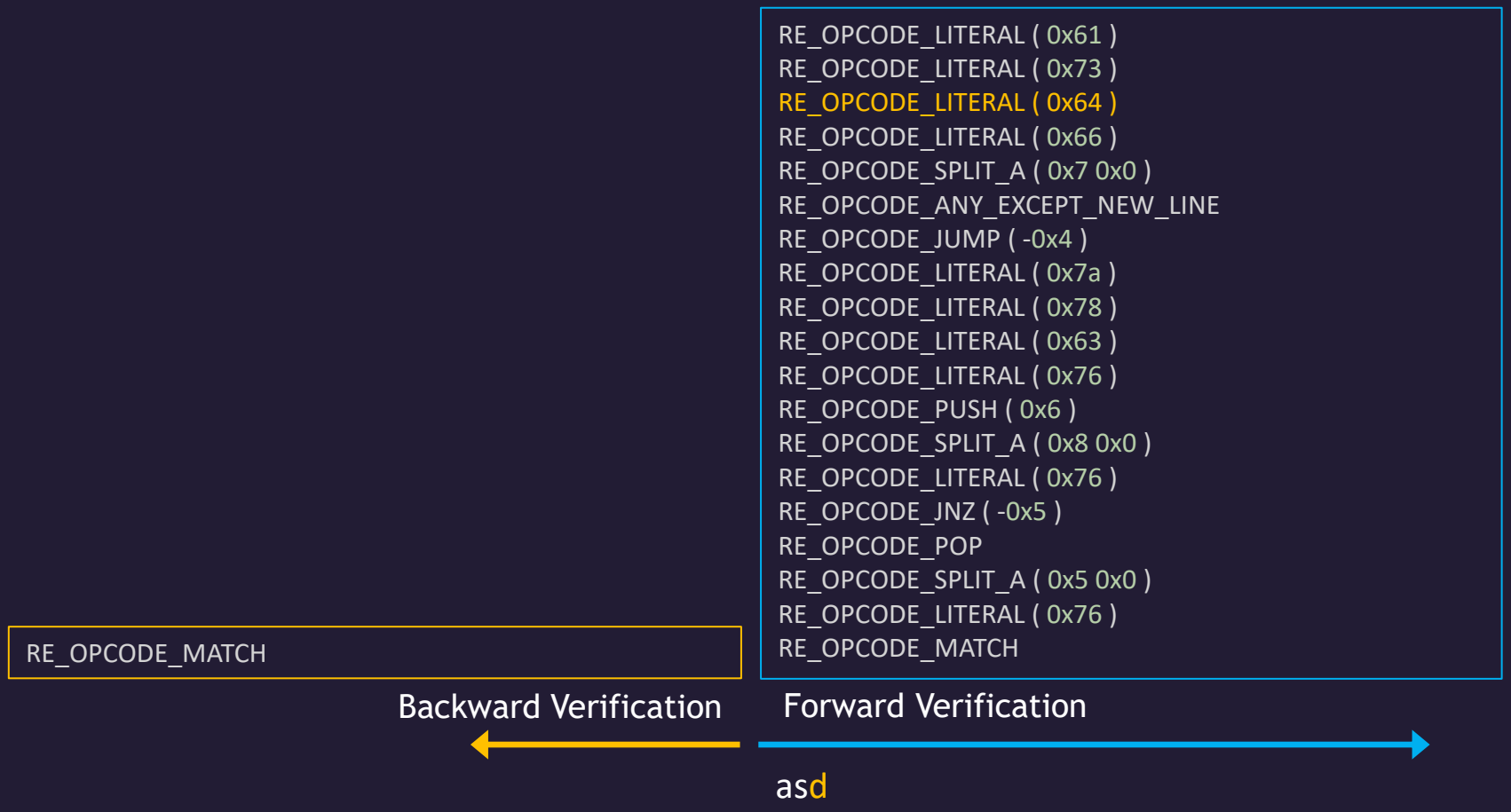
Decompiling Regular Expressions (v3.4.0)



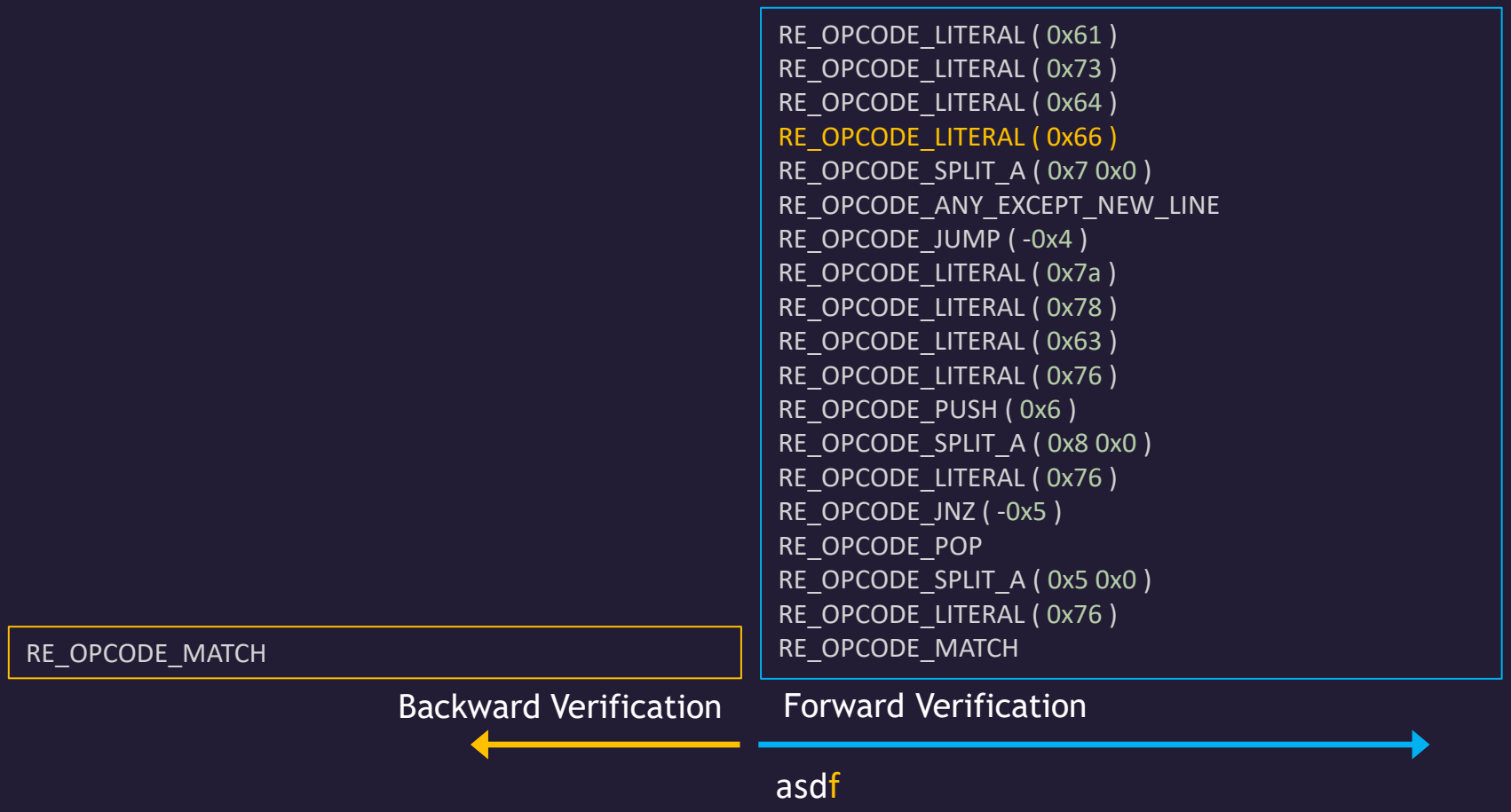
Decompiling Regular Expressions (v3.4.0)



Decompiling Regular Expressions (v3.4.0)



Decompiling Regular Expressions (v3.4.0)



Decompiling Regular Expressions (v3.4.0)

```
Code for e+ looks like:
  L1: code for e
      split L1, L2
  L2:
Code for e* looks like:
  L1: split L1, L2
      code for e
      jmp L1
  L2:
```

```
RE_OPCODE_LITERAL ( 0x61 )
RE_OPCODE_LITERAL ( 0x73 )
RE_OPCODE_LITERAL ( 0x64 )
RE_OPCODE_LITERAL ( 0x66 )
RE_OPCODE_SPLIT_A ( 0x7 0x0 )
RE_OPCODE_ANY_EXCEPT_NEW_LINE
RE_OPCODE_JUMP ( -0x4 )
RE_OPCODE_LITERAL ( 0x7a )
RE_OPCODE_LITERAL ( 0x78 )
RE_OPCODE_LITERAL ( 0x63 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_PUSH ( 0x6 )
RE_OPCODE_SPLIT_A ( 0x8 0x0 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_JNZ ( -0x5 )
RE_OPCODE_POP
RE_OPCODE_SPLIT_A ( 0x5 0x0 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_MATCH
```

```
RE_OPCODE_MATCH
```

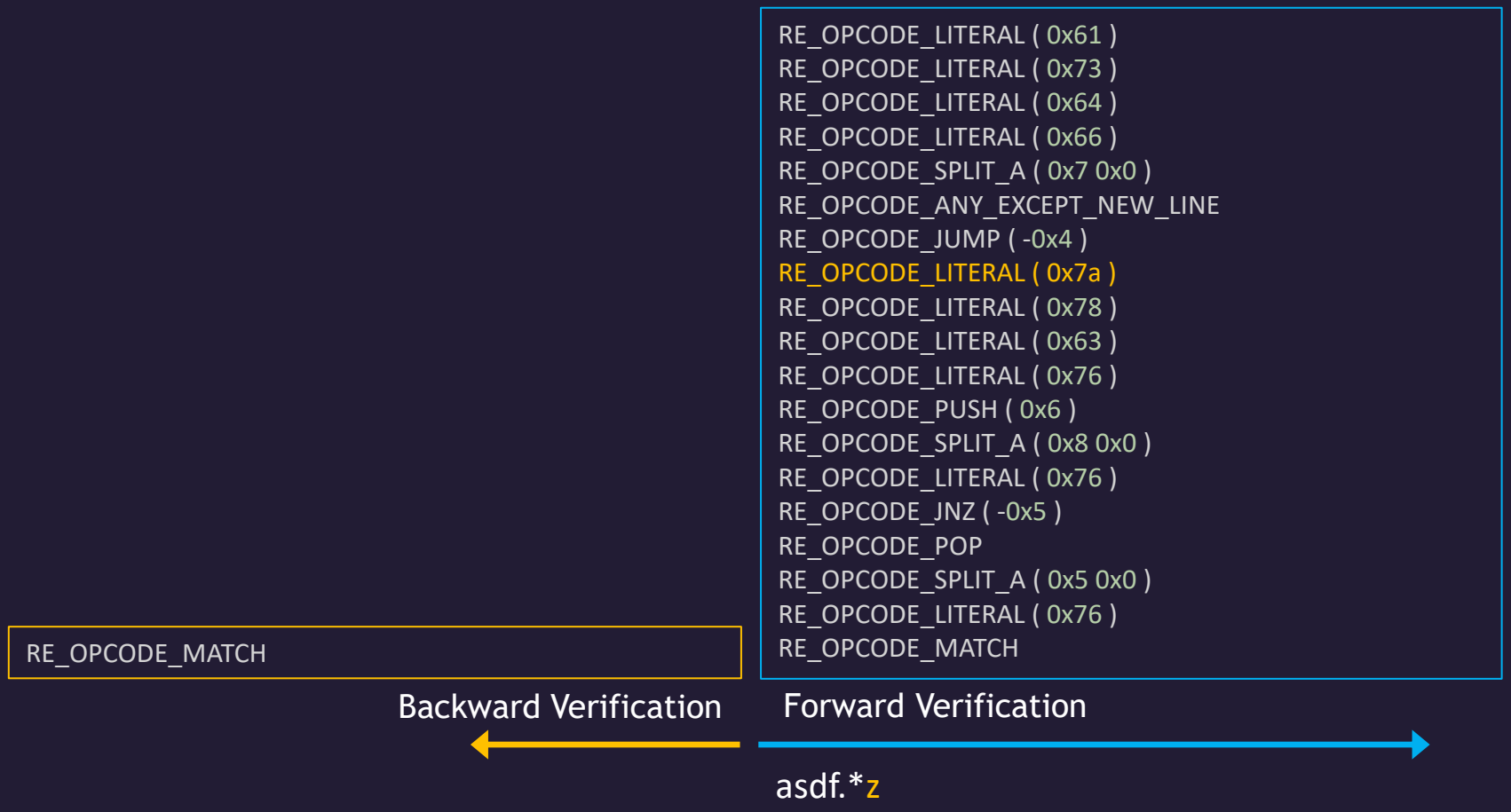
Backward Verification

Forward Verification

asdf.*

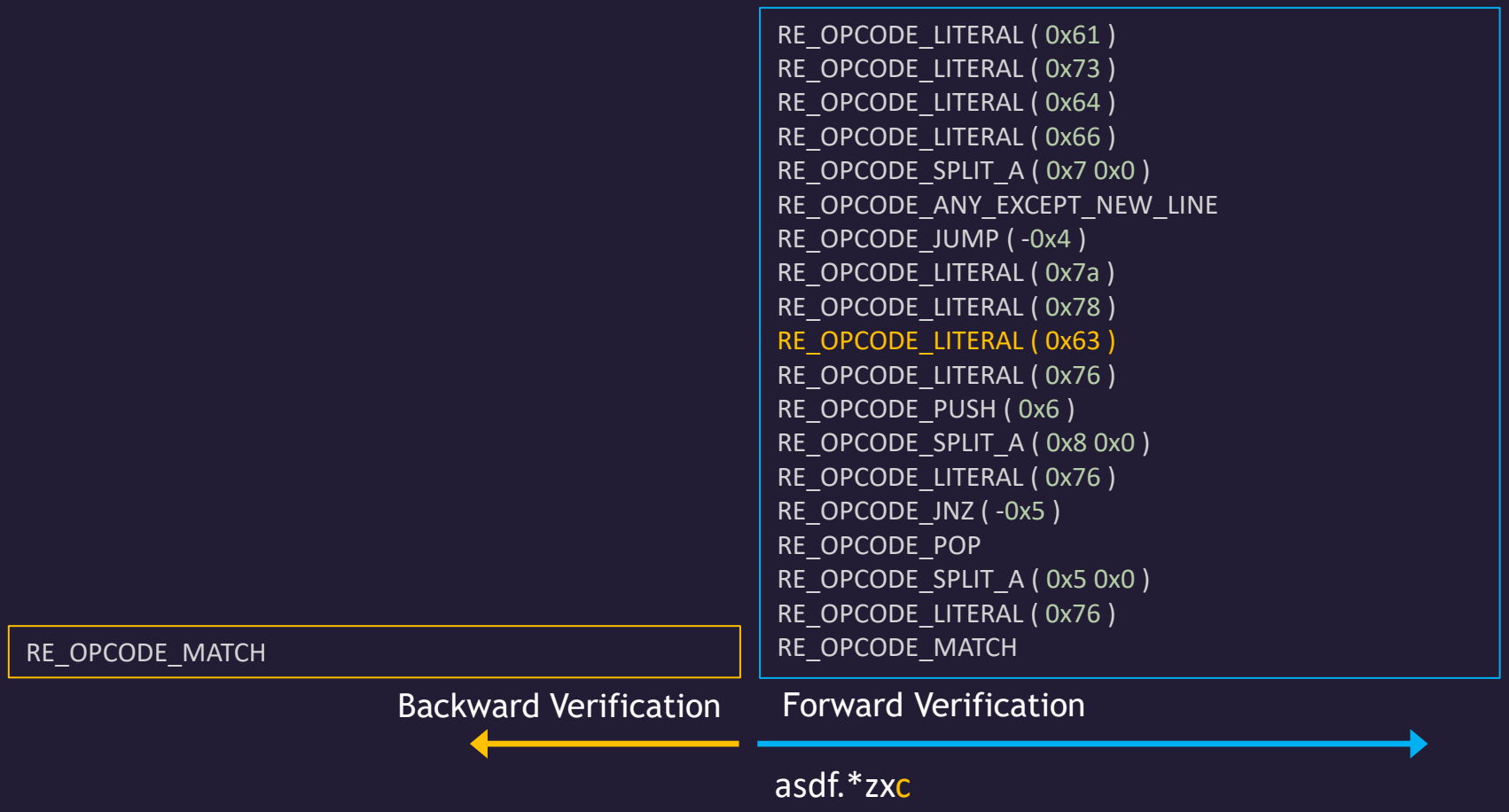
CYCRAFT

Decompiling Regular Expressions (v3.4.0)

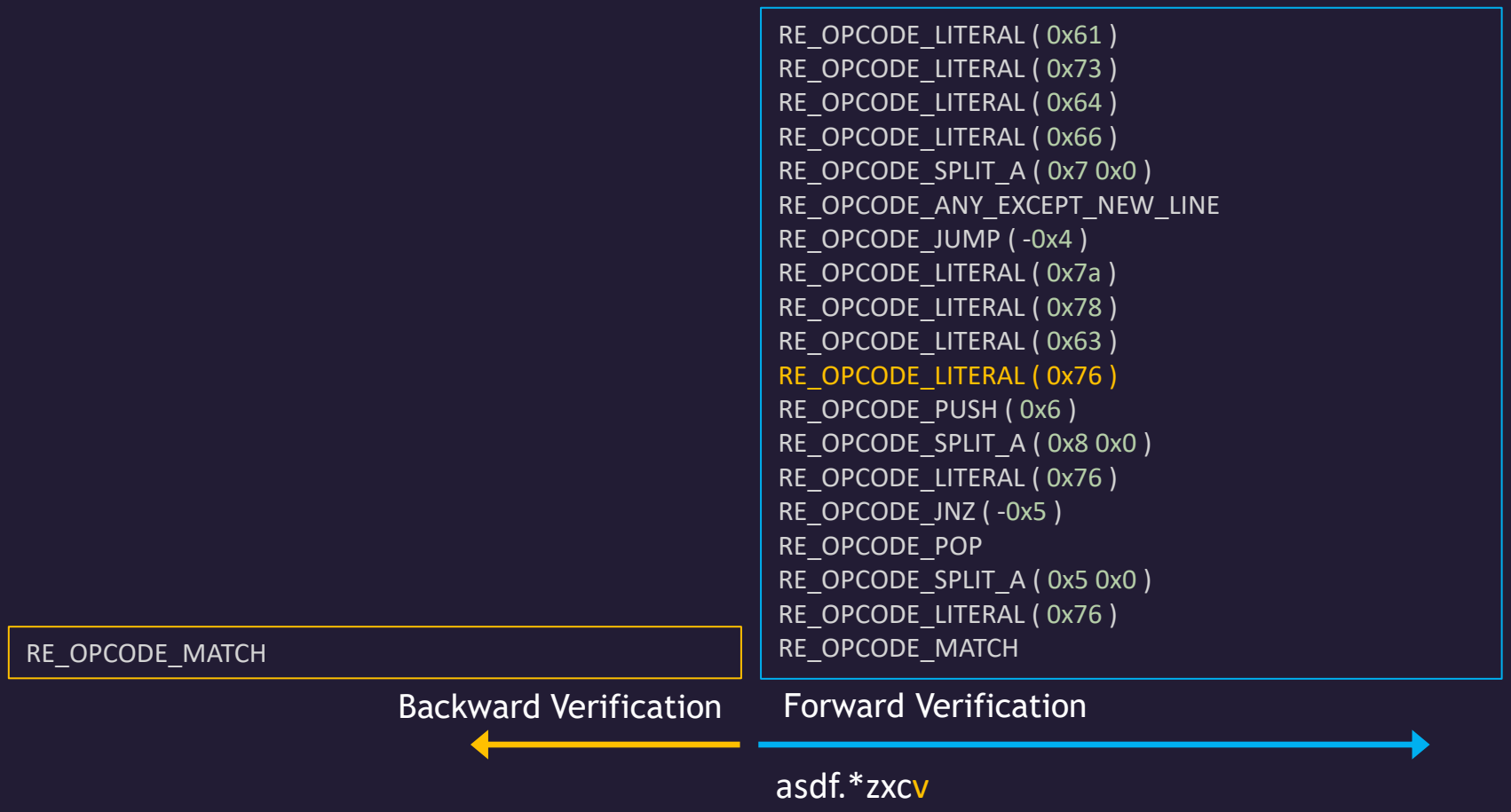


Decompiling Regular Expressions (v3.4.0)

Decompiling Regular Expressions (v3.4.0)



Decompiling Regular Expressions (v3.4.0)

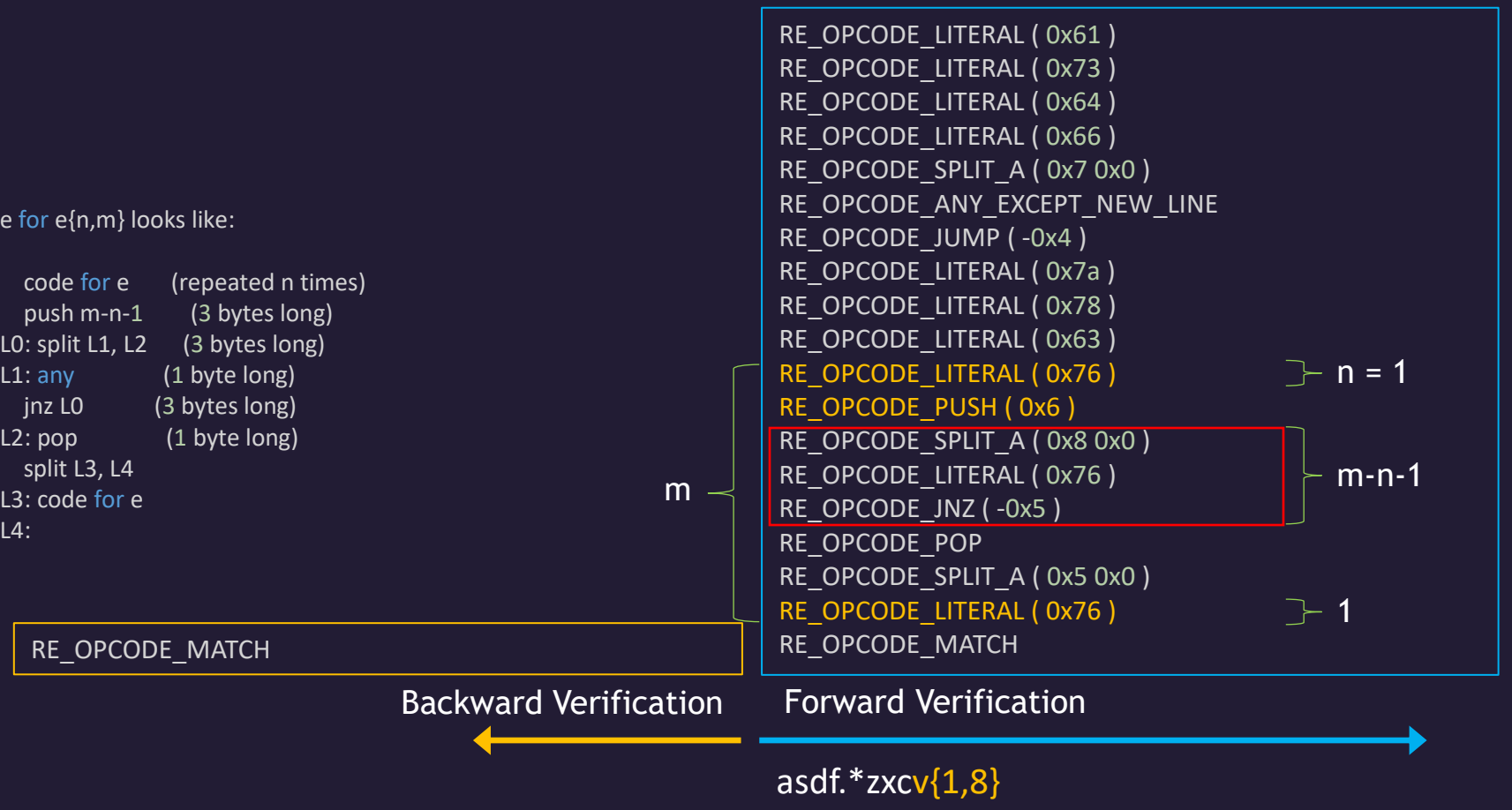


Decompiling Regular Expressions (v3.4.0)

Decompiling Regular Expressions (v3.4.0)

Code for e{n,m} looks like:

```
    code for e      (repeated n times)
    push m-n-1      (3 bytes long)
L0: split L1, L2    (3 bytes long)
L1: any             (1 byte long)
    jnz L0          (3 bytes long)
L2: pop             (1 byte long)
    split L3, L4
L3: code for e
L4:
```



m-n-1 = 0x6

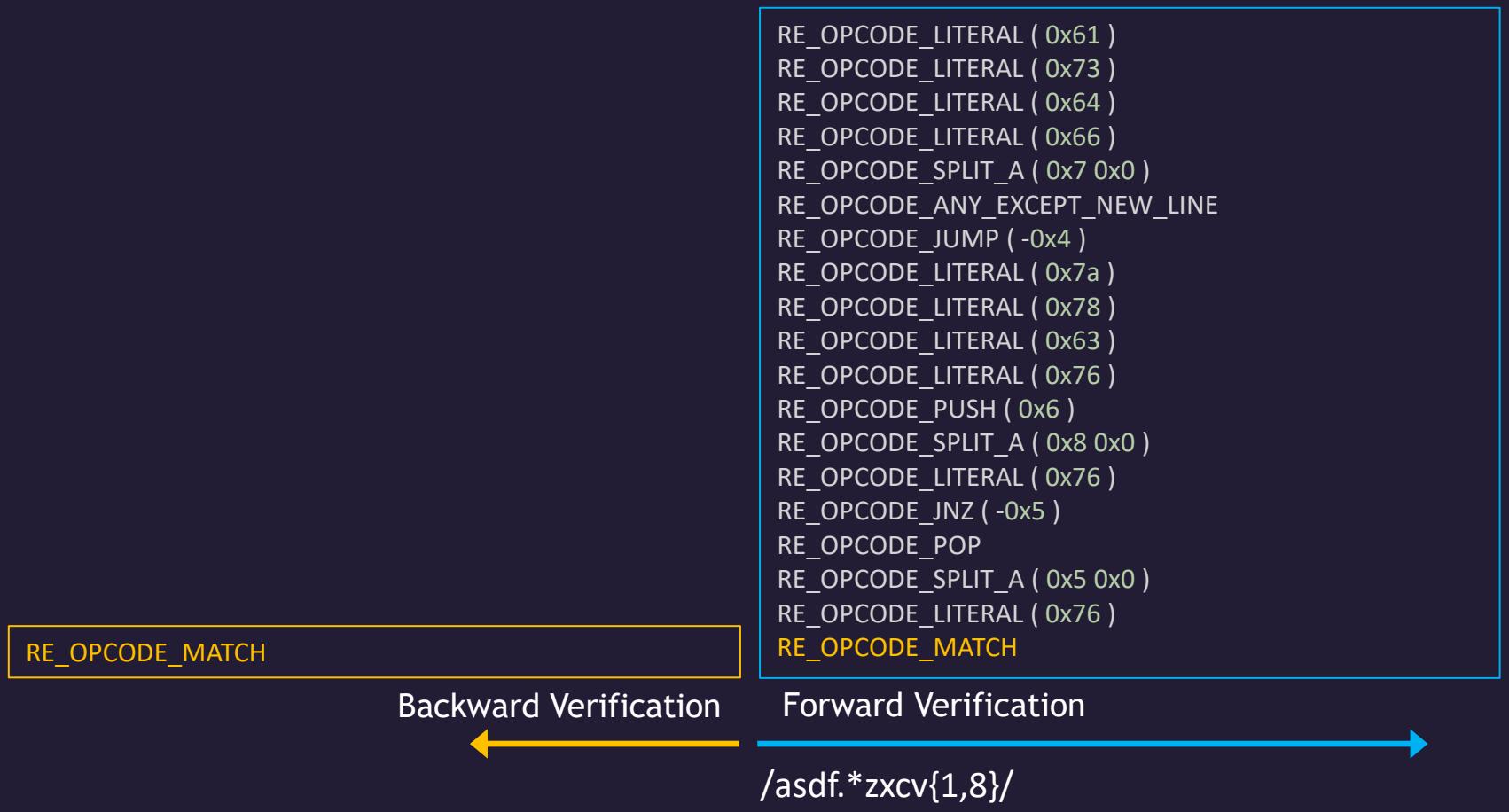
m = (m-n-1) + n + 1

= 6 + 1 + 1

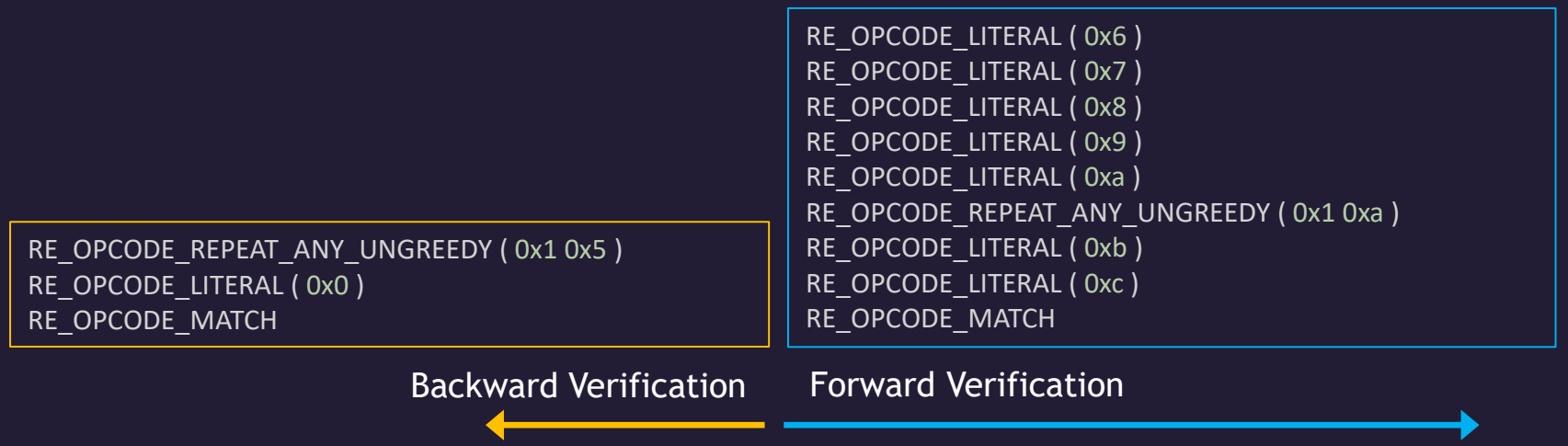
= 8

CYCRAFT

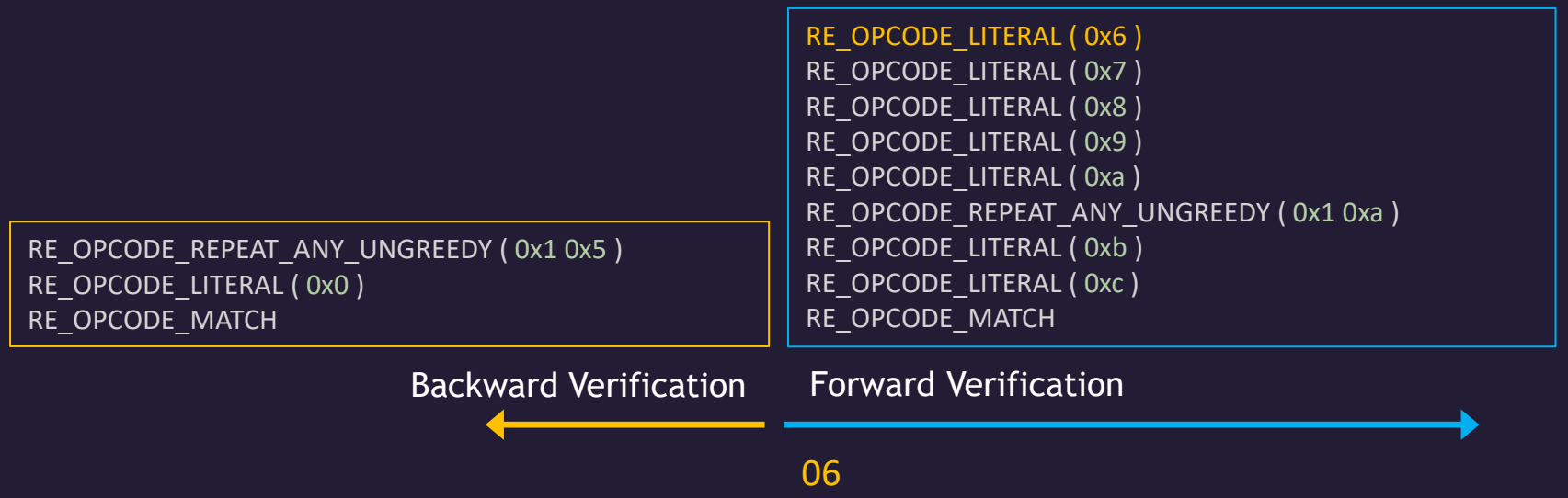
Decompiling Regular Expressions (v3.4.0)



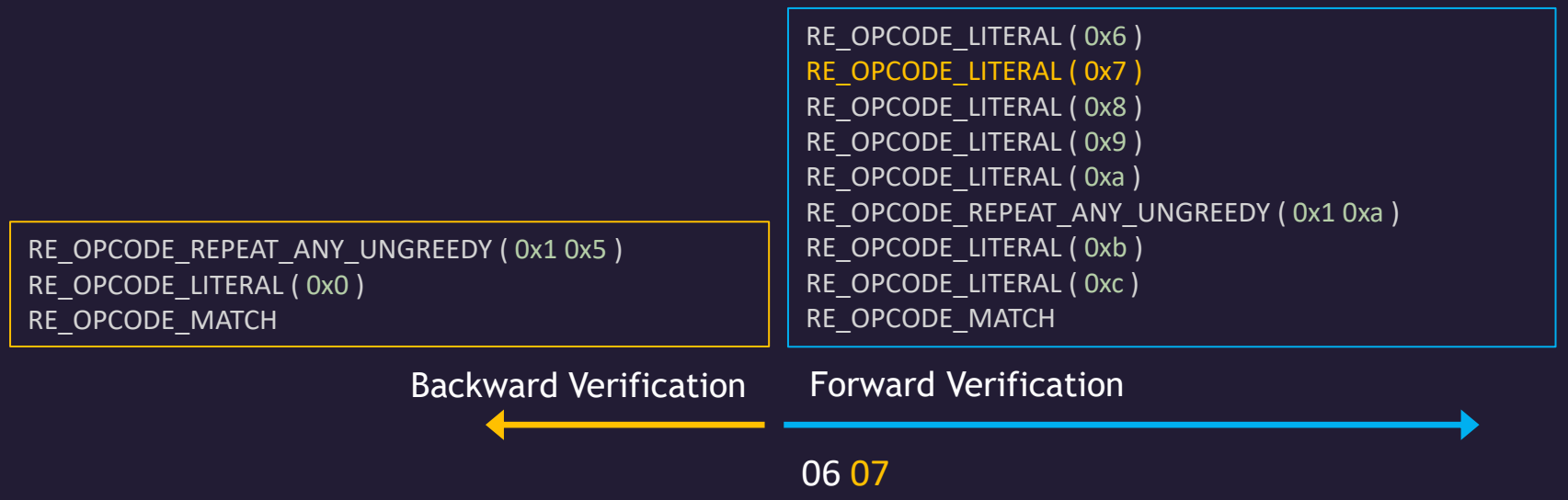
Decompiling Hex Patterns (v3.9.0)



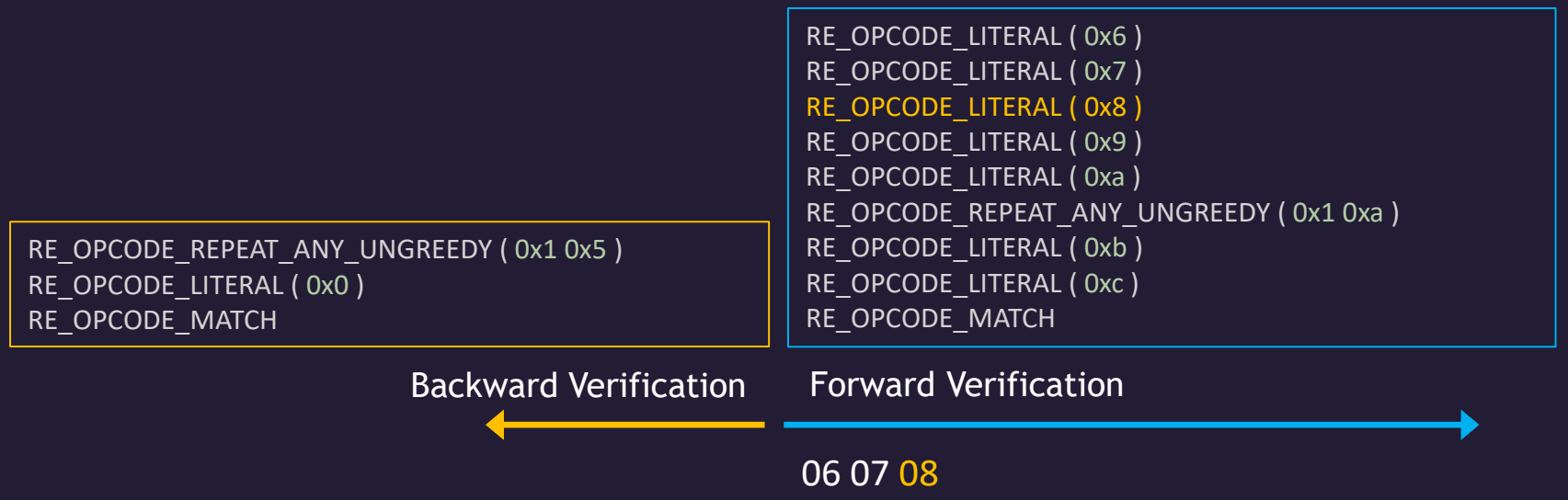
Decompiling Hex Patterns (v3.9.0)



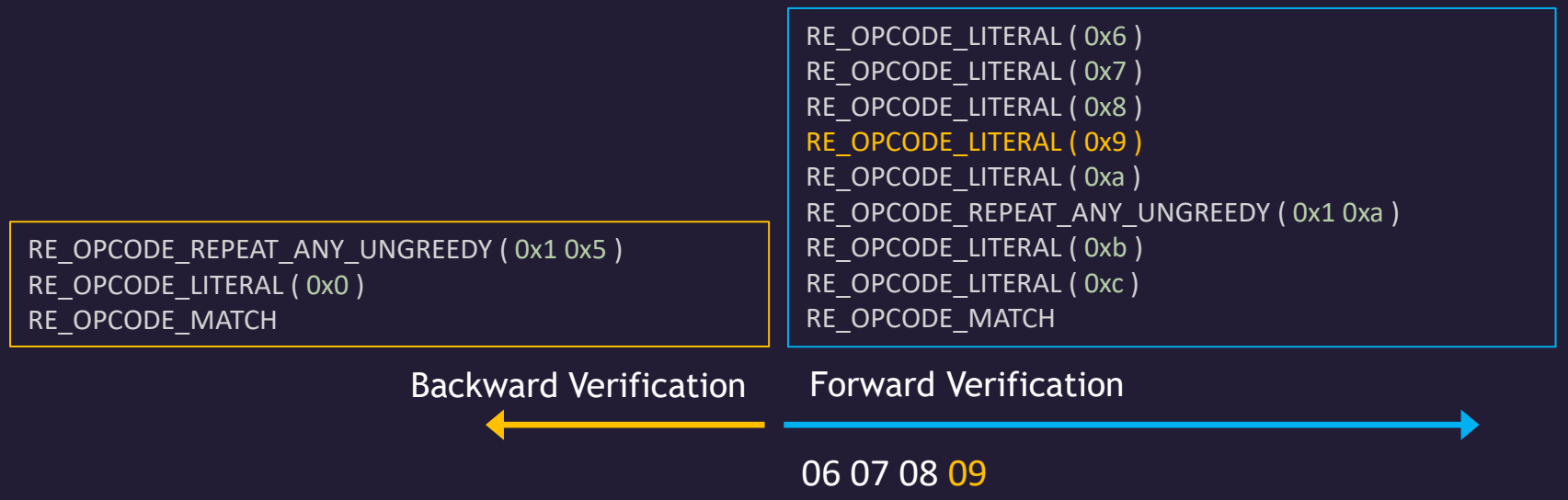
Decompiling Hex Patterns (v3.9.0)



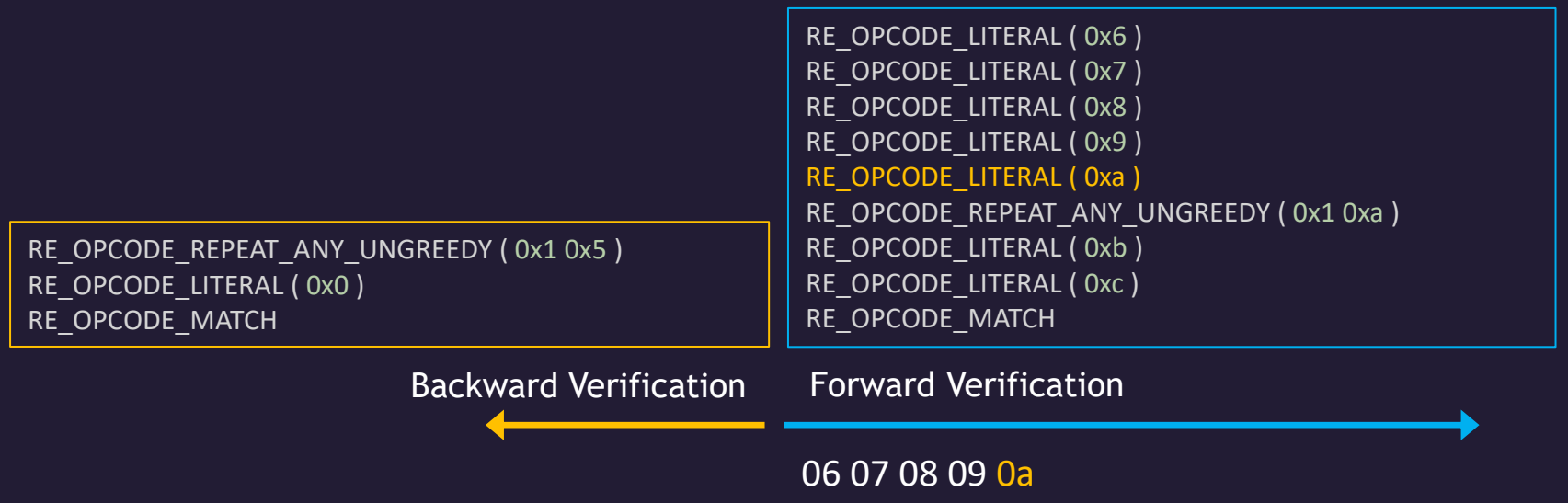
Decompiling Hex Patterns (v3.9.0)



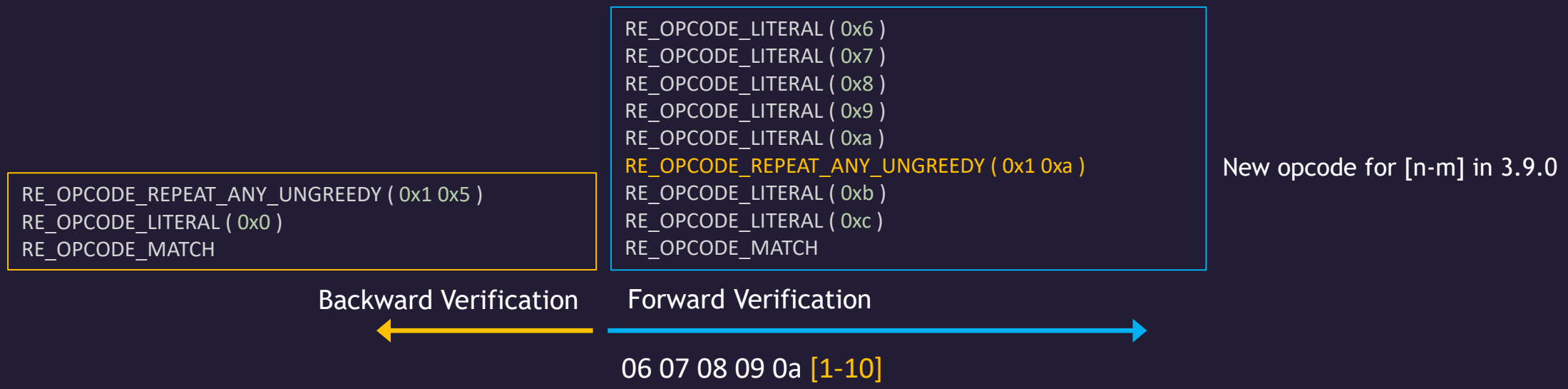
Decompiling Hex Patterns (v3.9.0)



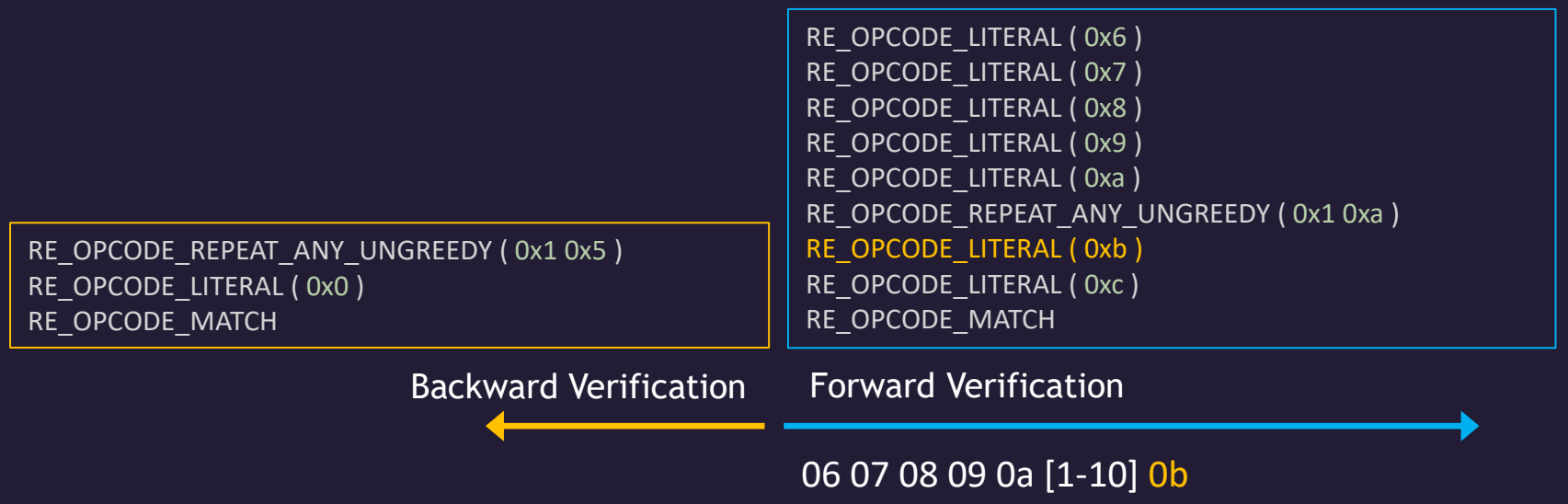
Decompiling Hex Patterns (v3.9.0)



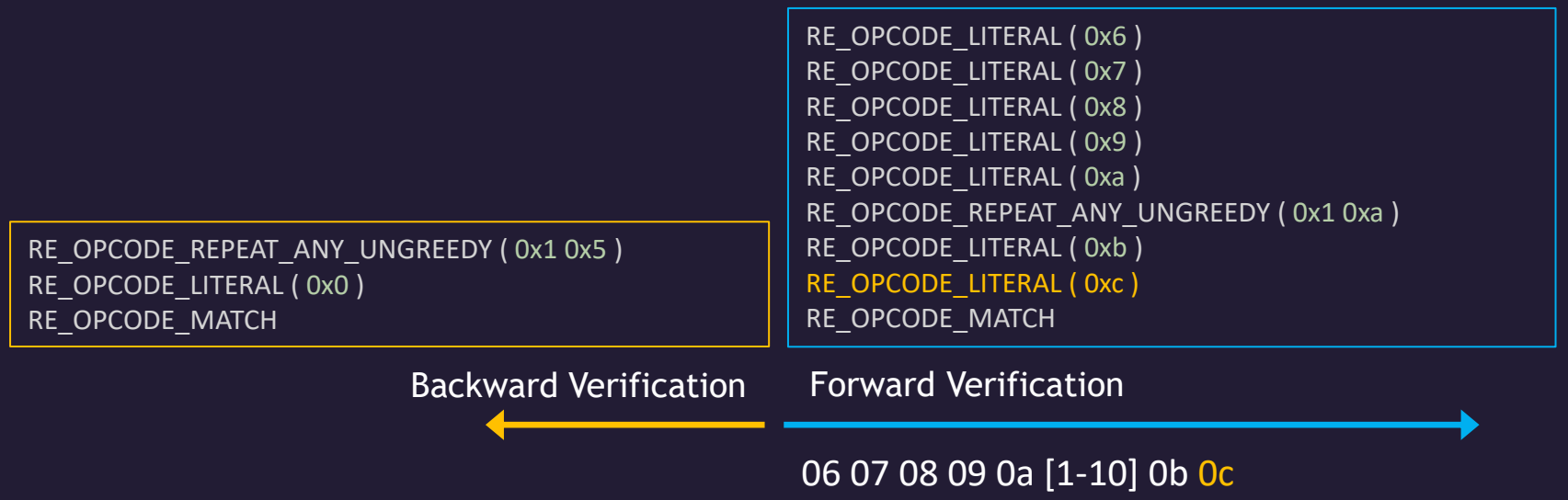
Decompiling Hex Patterns (v3.9.0)



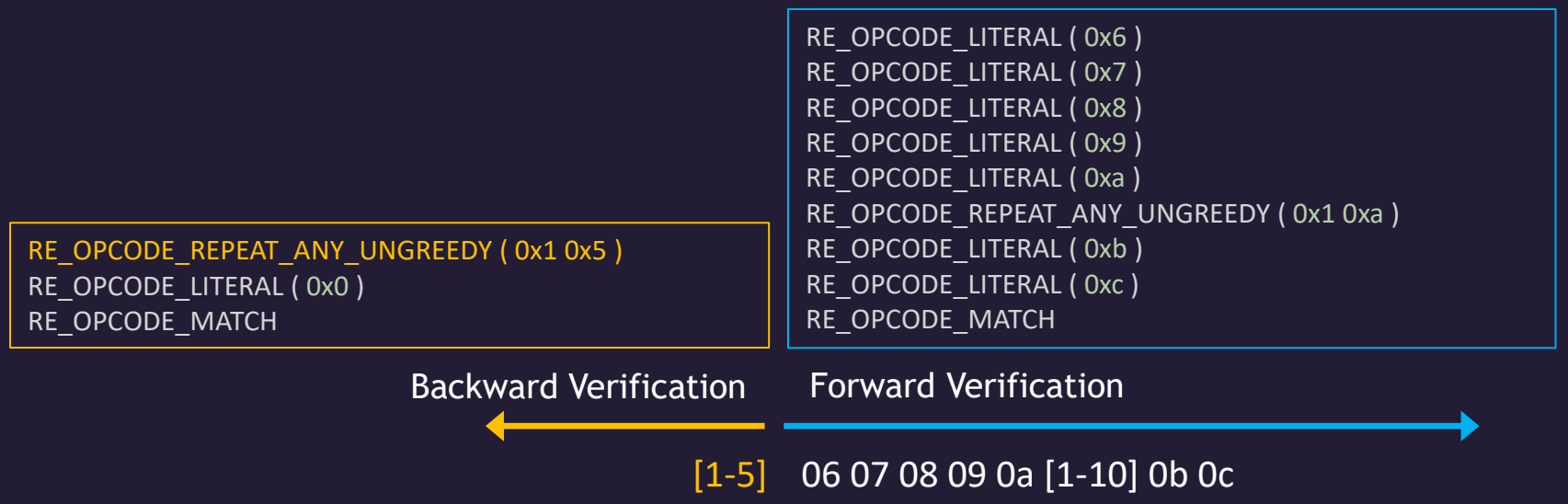
Decompiling Hex Patterns (v3.9.0)



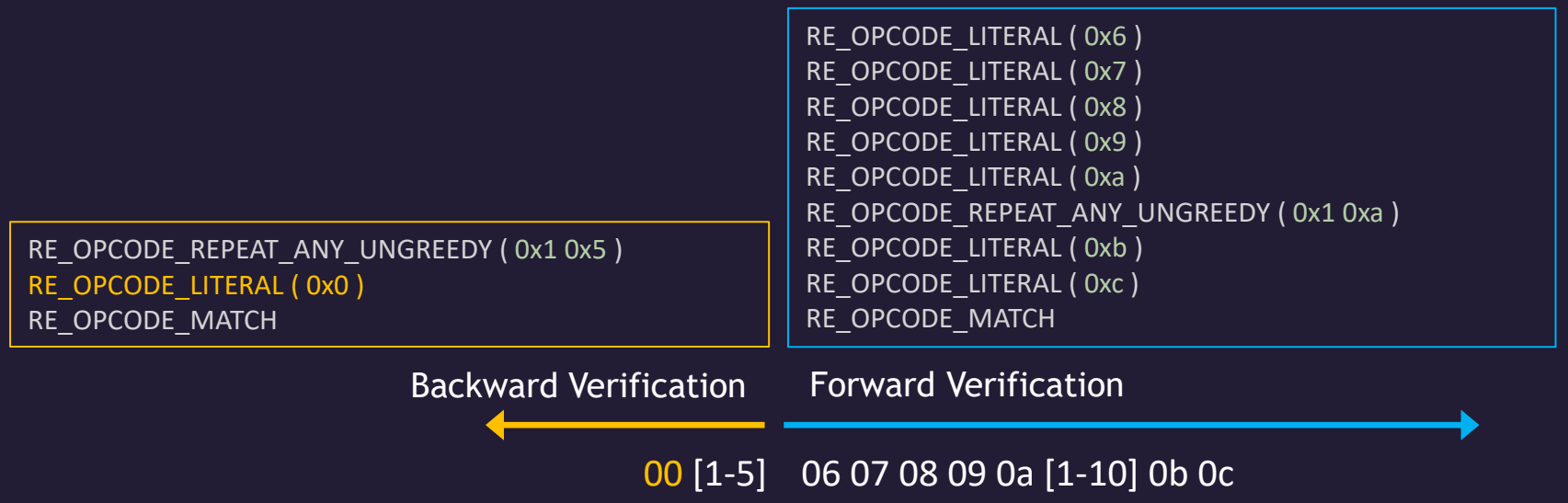
Decompiling Hex Patterns (v3.9.0)



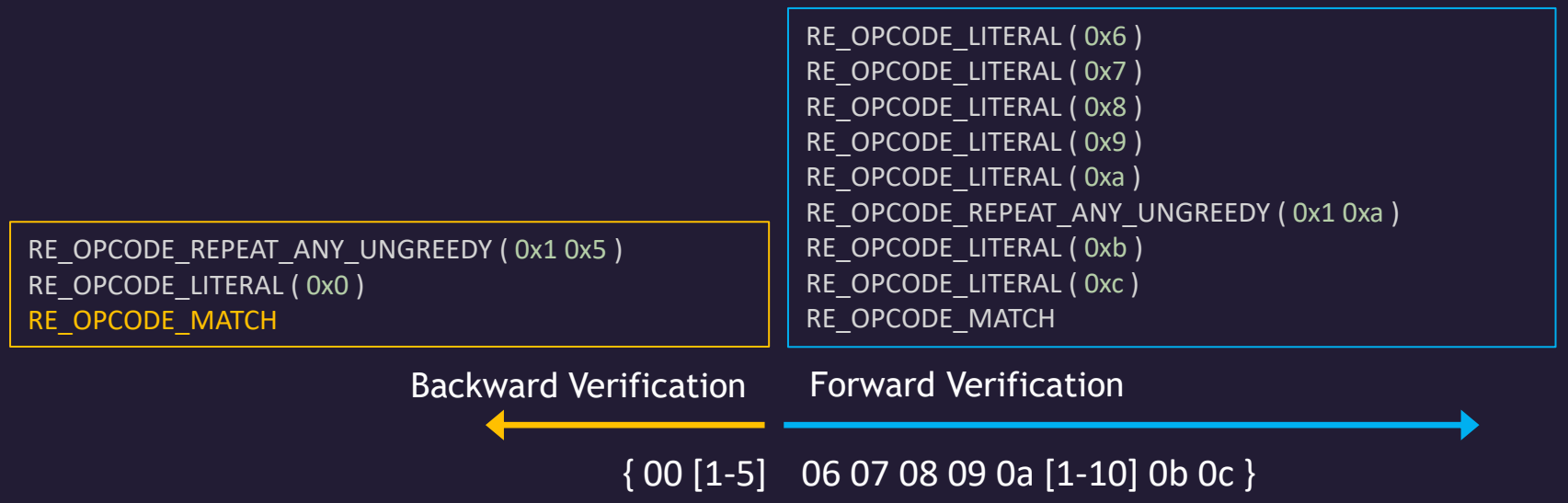
Decompiling Hex Patterns (v3.9.0)



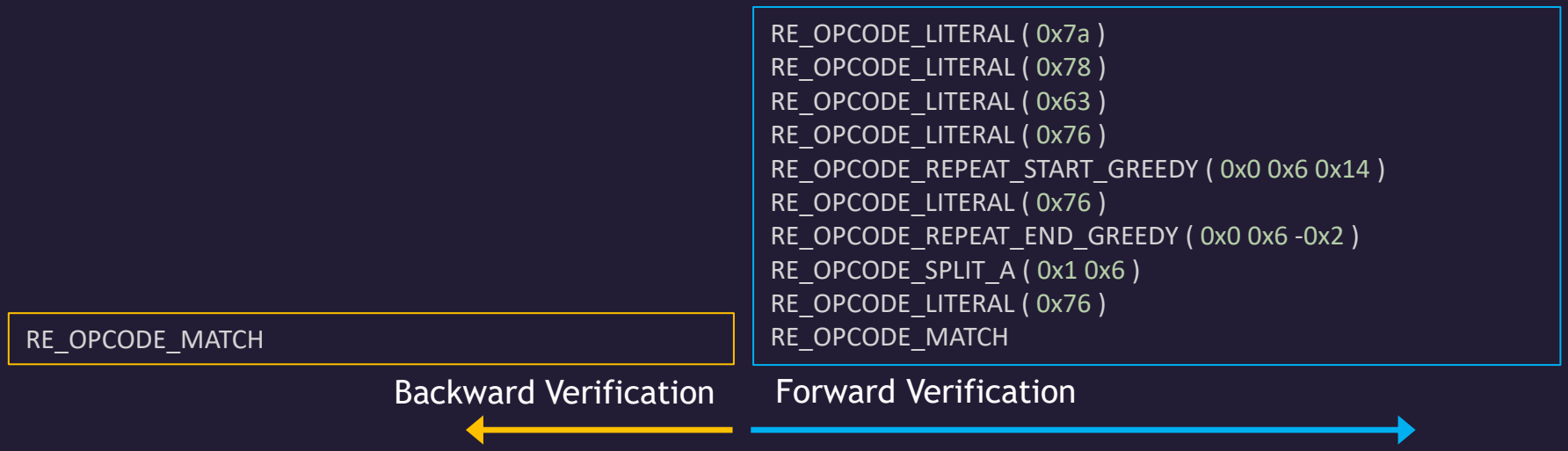
Decompiling Hex Patterns (v3.9.0)



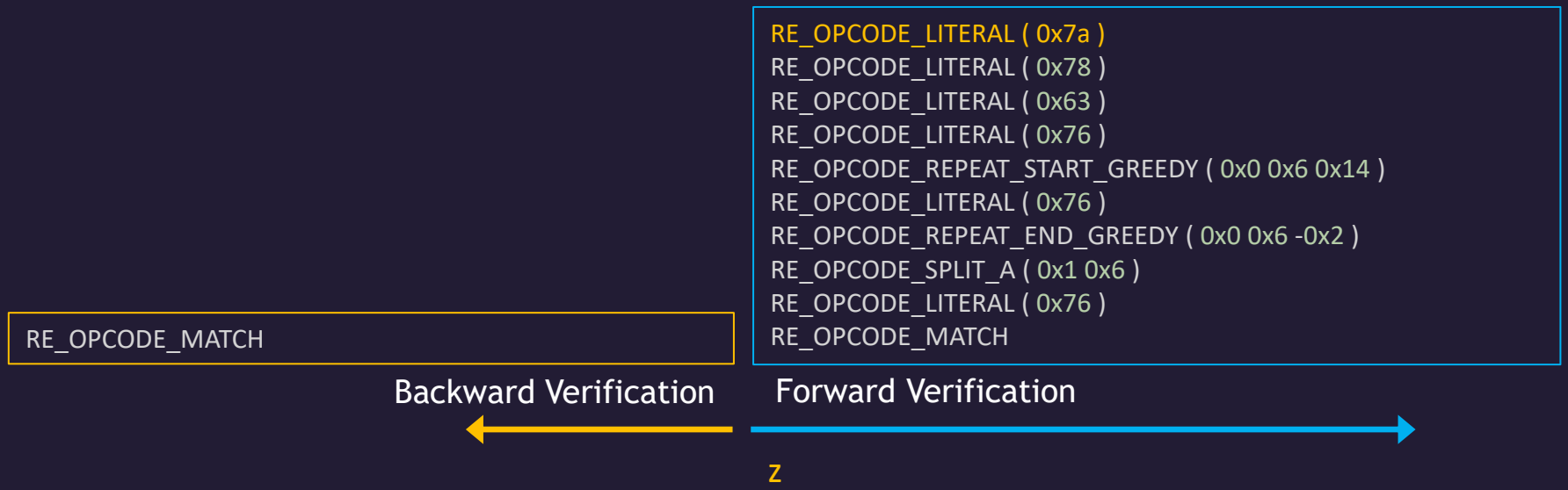
Decompiling Hex Patterns (v3.9.0)



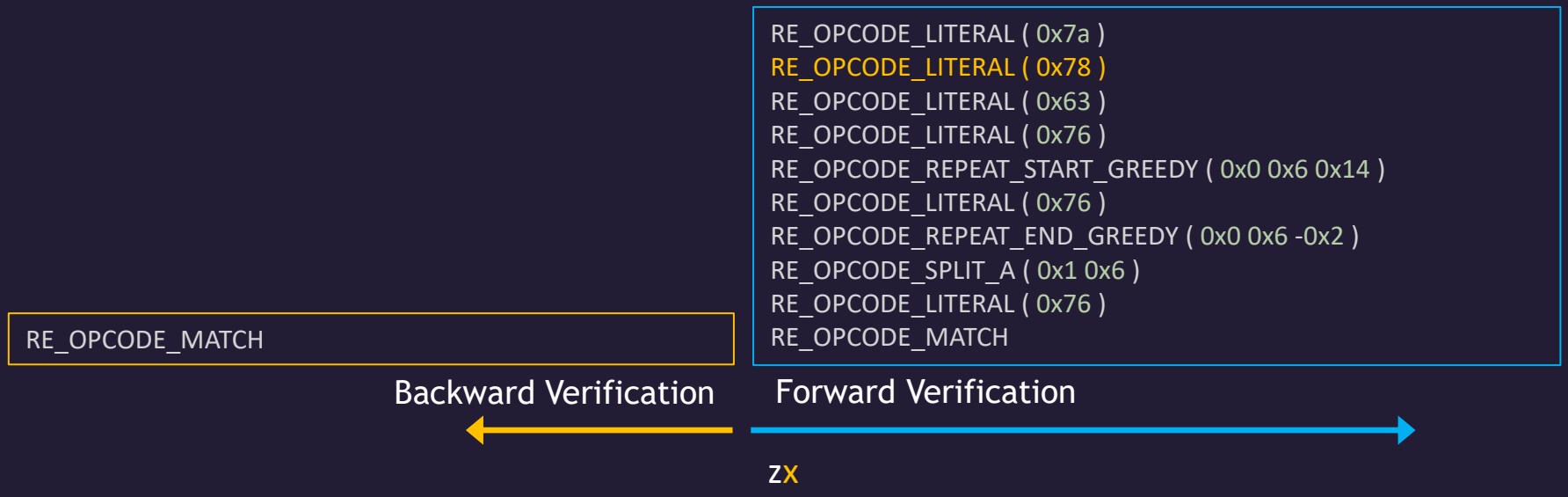
Decompiling Regular Expressions (v3.9.0)



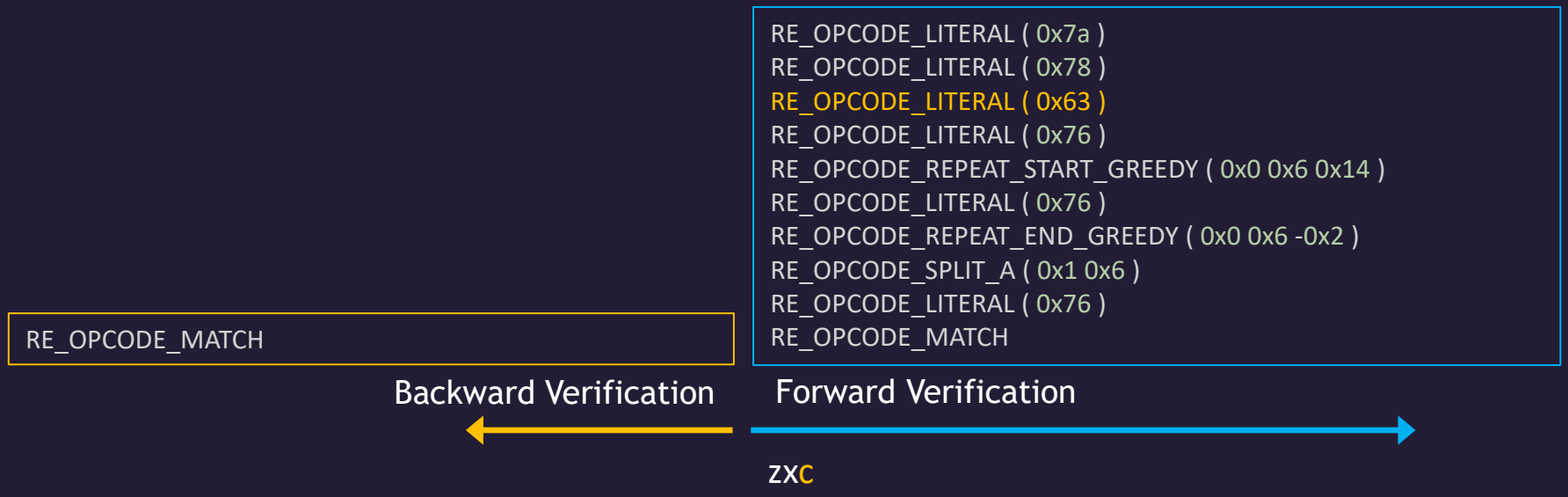
Decompiling Regular Expressions (v3.9.0)



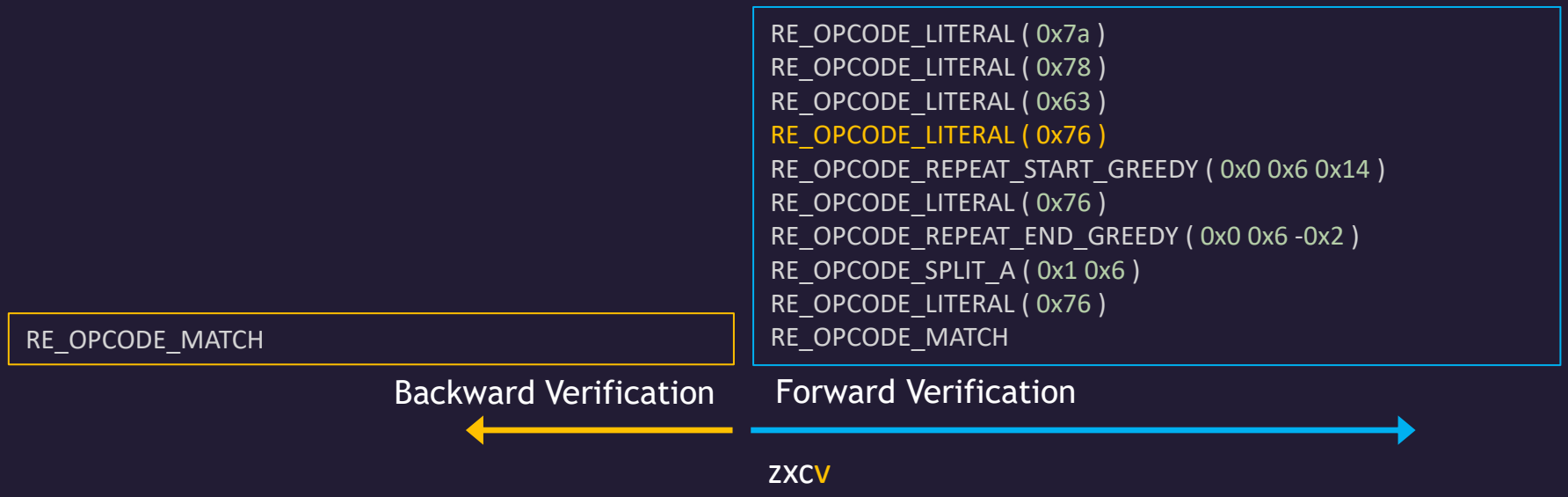
Decompiling Regular Expressions (v3.9.0)



Decompiling Regular Expressions (v3.9.0)



Decompiling Regular Expressions (v3.9.0)



Decompiling Regular Expressions (v3.9.0)

Code for e{n,m} looks like:

```
    code for e          ---   prolog
    repeat_start min, max, L1   --+
L0: code for e          |   repeat
    repeat_end min, max, L0     --+
L1: split L2, L3        ---   split
L2: code for e          ---   epilog
L3:
```

Not all sections (prolog, repeat, split and epilog) are generated in all cases, it depends on the values of n and m. The table shows which sections are generated for the first few values of n and m.

n,m	prolog	repeat (min,max)	split	epilog
0,0	-	-	-	-
0,1	-	-	X	X
0,2	-	0,1	X	X
0,3	-	0,2	X	X
0,M	-	0,M-1	X	X
1,1	X	-	-	-
1,2	X	-	X	X
1,3	X	0,1	X	X
1,4	X	1,2	X	X
1,M	X	1,M-2	X	X

n = min + (prolog?) + (!split?)

= 0 + 1 + 0 = 1

m = max + 1 + (prolog?)

= 6 + 1 + 1 = 8

```
RE_OPCODE_LITERAL ( 0x7a )
RE_OPCODE_LITERAL ( 0x78 )
RE_OPCODE_LITERAL ( 0x63 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_REPEAT_START_GREEDY ( 0x0 0x6 0x14 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_REPEAT_END_GREEDY ( 0x0 0x6 -0x2 )
RE_OPCODE_SPLIT_A ( 0x1 0x6 )
RE_OPCODE_LITERAL ( 0x76 )
RE_OPCODE_MATCH
```

prolog

repeat

epilog

```
RE_OPCODE_MATCH
```

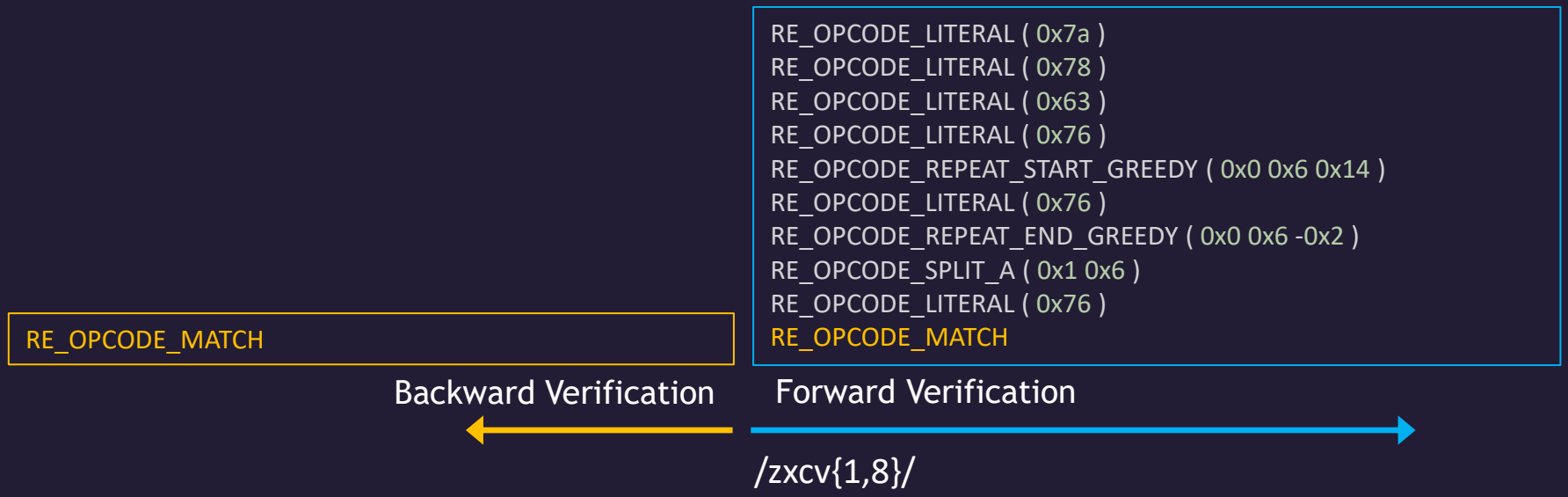
Backward Verification

Forward Verification

zxcv{1,8}

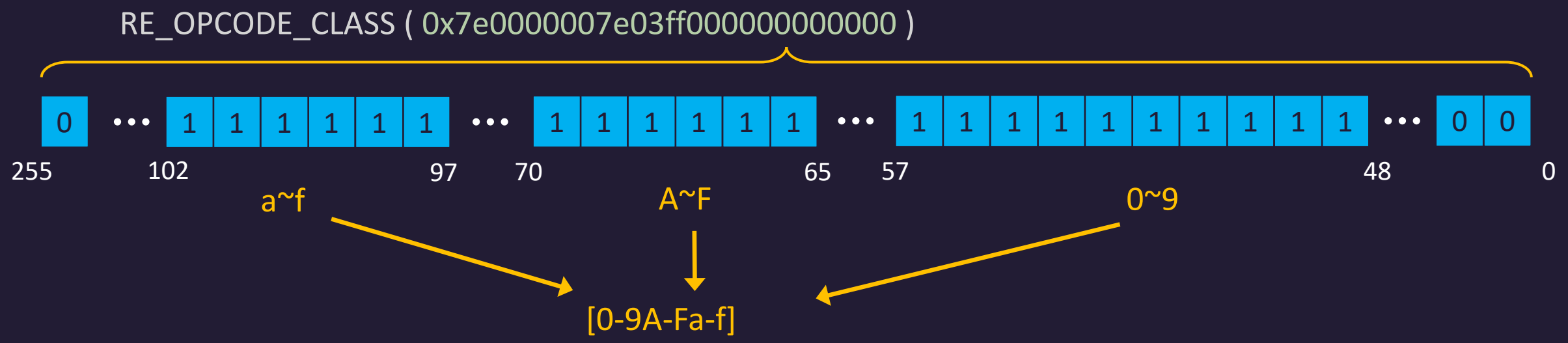
CYCRAFT

Decompiling Regular Expressions (v3.9.0)



Decompiling Regular Expressions

- Two opcodes for CLASS structure ([pattern])
  - RE_OPCODE_CLASS
  - RE_OPCODE_CLASS_NOCASE
- A 256-bit argument for representing ASCII characters



Evaluation

Methodology

- Yara-Rules/rules is used for testing
- 12429 rules of 10 categories are compiled with Yara v3.9
- We use YaDa to decompile it and see how many rules we could successfully recover

Results

Category	#rules	#success	#failes	Success rate
antidebug	55	55	0	100%
capabilities	52	52	0	100%
crypto	122	122	0	100%
Common CVEs	19	19	0	100%
email	18	18	0	100%
exploit kits	74	74	0	100%
maldocs	71	58	13	81.69%
malware	2065	1983	82	96.02%
packers	9315	290	9025	3.11%
webshells	638	637	1	99.84%
Total	**12429**	3308	9121	26.61%

Discussion

- Only 3.11% success on packer category
  - They used "pe" external module, which YaDa does not support yet
- 81.69%, 96.02% & 99.04% on maldocs, malware and webshells
  - Memory instructions, which YaDa does not support yet
  - Complex nested regex structure, which YaDa decompilation algorithm failed
- If we excluded rules with external module, the overall success rate is 86.6%
- We also tested on some non-public yara rules, where achieve a recover rate of ~85%

Future Directions (perhaps...)

- Support external modules
- Support decompilation of memory instructions
- Support more versions
- Re-write in faster language...
- Craft malicious bytecode to exploit yara

Special Thanks

- Jean-Baptiste Galet (https://jbgalet.fr/)
- Inndy Lin (https://www.inndy.tw/)

</slide>