

# Breaking State-of-the-Art Binary Code Obfuscation

## A Program Synthesis-based Approach

REcon Brussels

February 2, 2018

---

Tim Blazytko, @mr\_phrazer  
<http://synthesis.to>

Moritz Contag, @dwuid  
<https://dwuid.com>

Chair for Systems Security  
Ruhr-Universität Bochum  
`<firstname.lastname>@rub.de`



# Syntia: Synthesizing the Semantics of Obfuscated Code

Tim Blazytko, Moritz Contag, Cornelius Aschermann,  
and Thorsten Holz, *Ruhr-Universität Bochum*

<https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/blazytko>

This paper is included in the Proceedings of the  
26th USENIX Security Symposium  
August 16–18, 2017 • Vancouver, BC, Canada

ISBN 978-1-931971-40-9

- ?
- Obfuscated code, semantics?
  
- ?
- Traditional deobfuscation techniques
  
- Orthogonal approach

Prevent Complicate reverse engineering attempts.

- Intellectual Property
- Malicious Payloads
- Digital Rights Management

Prevent Complicate reverse engineering attempts.

- Intellectual Property
- Malicious Payloads
- Digital Rights Management

“We achieved our goals. We were uncracked for **13 whole days**.”

– Martin Slater, 2K Australia, on *BioShock* (2007).

How to protect software?

Abuse shortcomings of file parsers and other tools of the trade.

- `fld tbyte ptr [__bad_values]` crashing OllyDbg 1.10.
- Fake `SizeOfImage` crashing process dumpers.

Abuse shortcomings of file parsers and other tools of the trade.

- `fld tbyte ptr [__bad_values]` crashing OllyDbg 1.10.
- Fake `SizeOfImage` crashing process dumpers.

Detect artifacts of the debugging process.

- `PEB.BeingDebugged` bit being set.
- `int 2D` and exception handling in debuggers.

Abus

game does not start debugger detected

All Videos Shopping Images News More Settings Tools

About 6.370.000 results (0,51 seconds)

[When i run this game i get a debugger error message Debugger ...](https://support.ubi.com/.../When-i-run-this-game-i-get-a-debugger-error-message-De...)  
[https://support.ubi.com/.../When-i-run-this-game-i-get-a-debugger-error-message-De... ▾](https://support.ubi.com/.../When-i-run-this-game-i-get-a-debugger-error-message-De...)  
When i run this game i get the following error message : **Debugger Detected** - Please close it down and restart! Windows NT ... Our game will not run while this application is running in memory, to stop this from happening you will need to stop MDM.exe as a startup process. Do the following : Goto the "Start" button --> "Run".

Dete

1. We want the technique to be *semantics-preserving*.

Preserve the observable behavior of the application.

1. We want the technique to be *semantics-preserving*.
2. We want to avoid external dependencies, focus on code only.

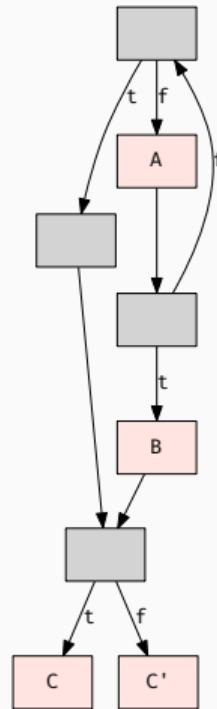
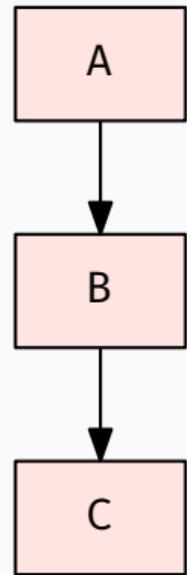
Assume white-box attack scenario.

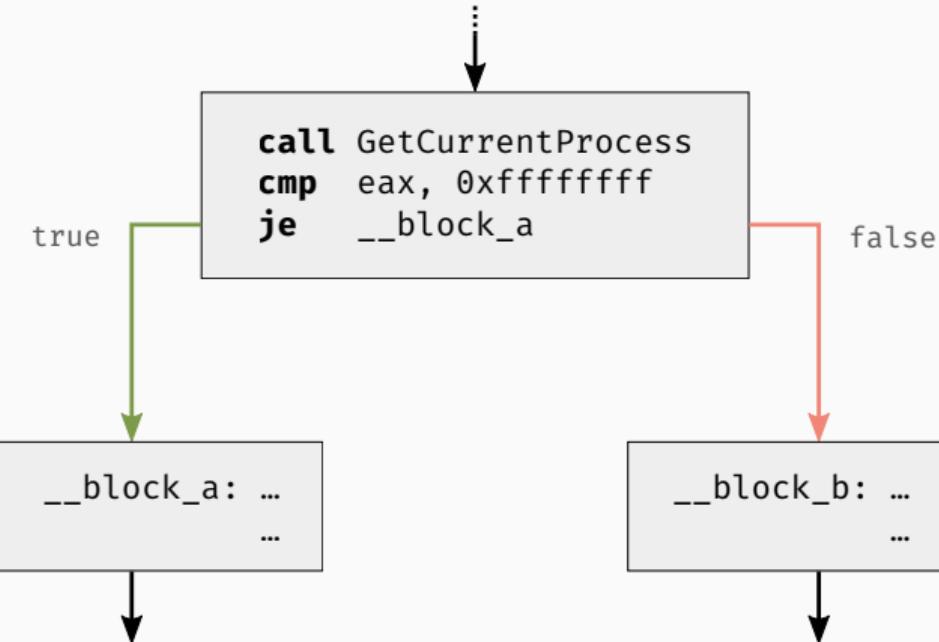
1. We want the technique to be *semantics-preserving*.
2. We want to avoid external dependencies, focus on code only.
3. We want techniques where  $\text{effort}(\text{deploy}) \ll \text{effort}(\text{attack})$ .

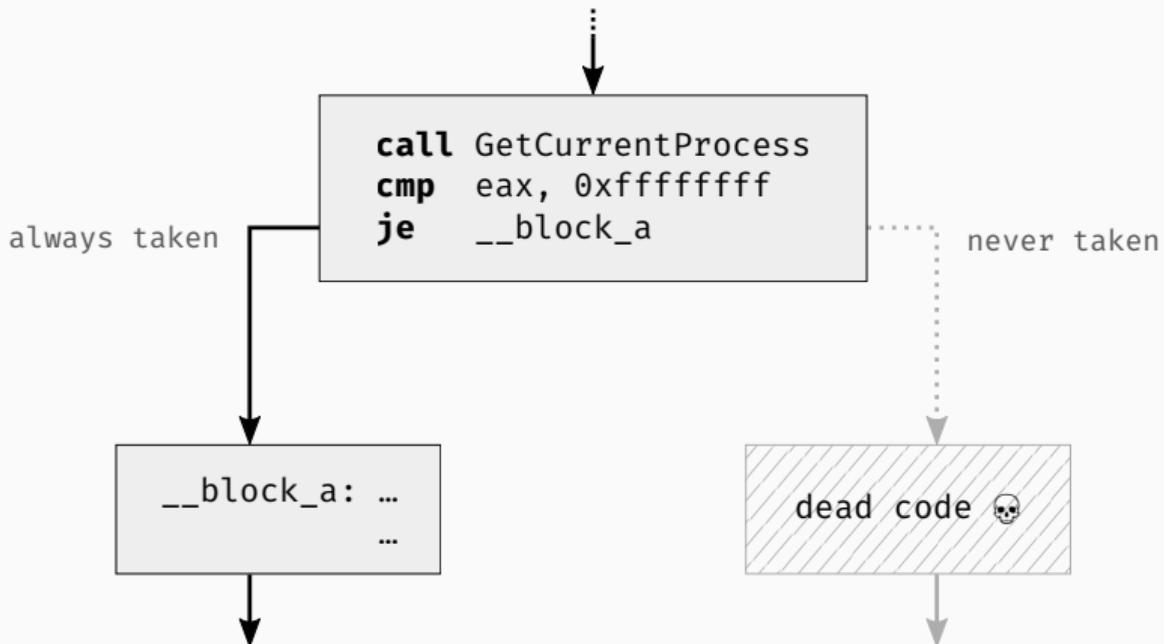
Anti-Debugging tricks are effort 1:1.

# Code Obfuscation Techniques

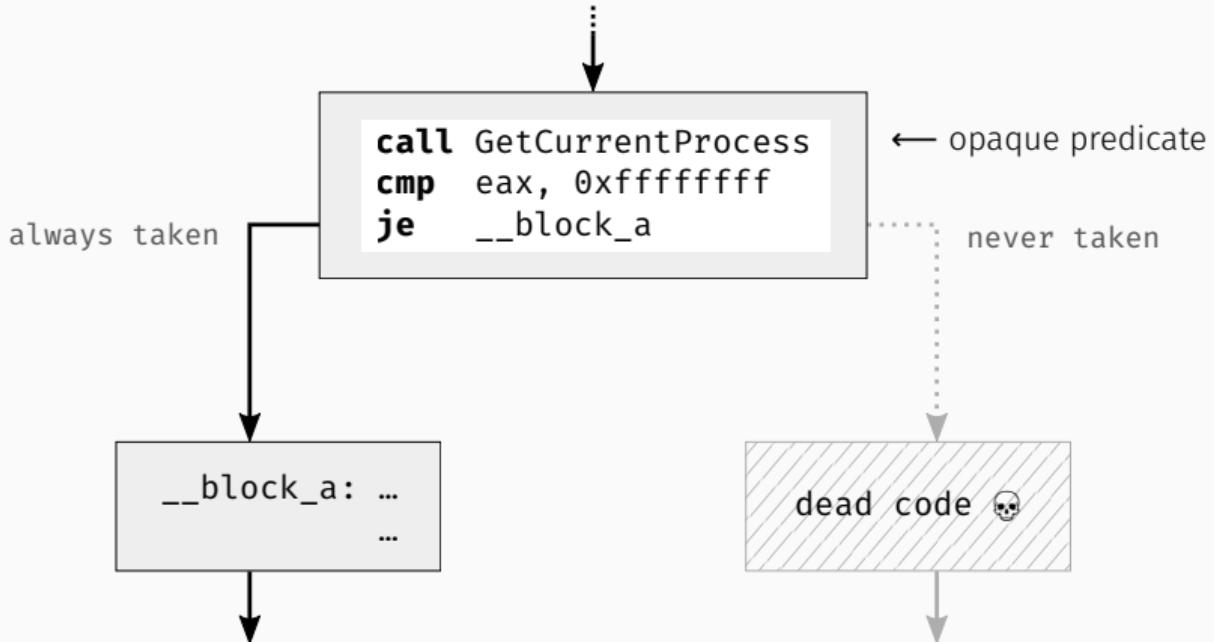
Opaque Predicates



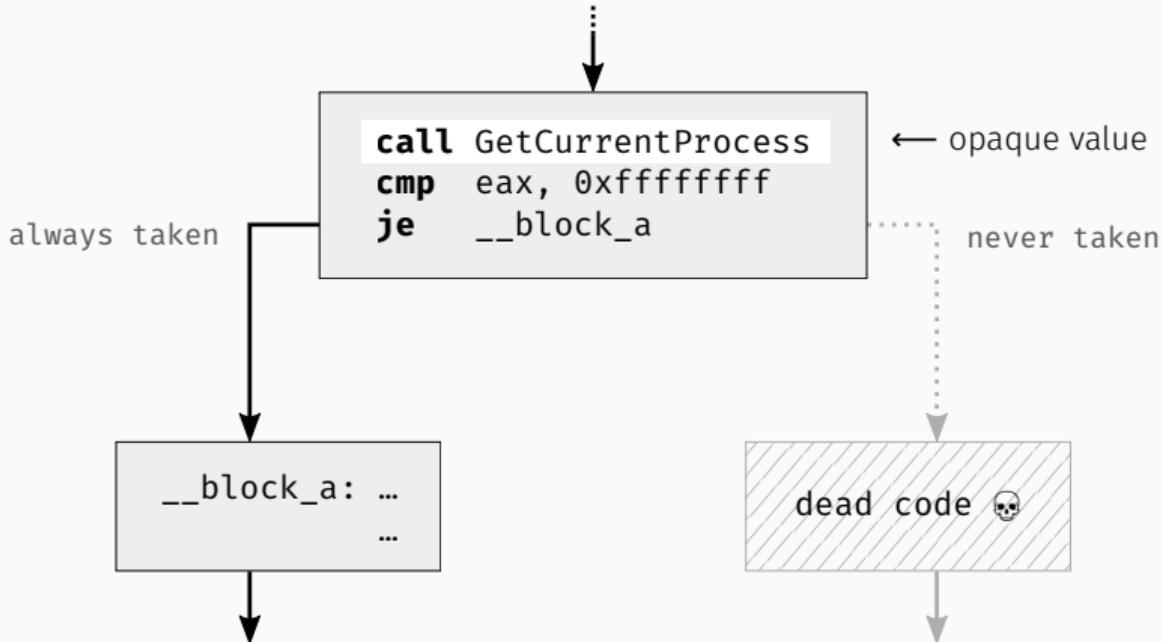




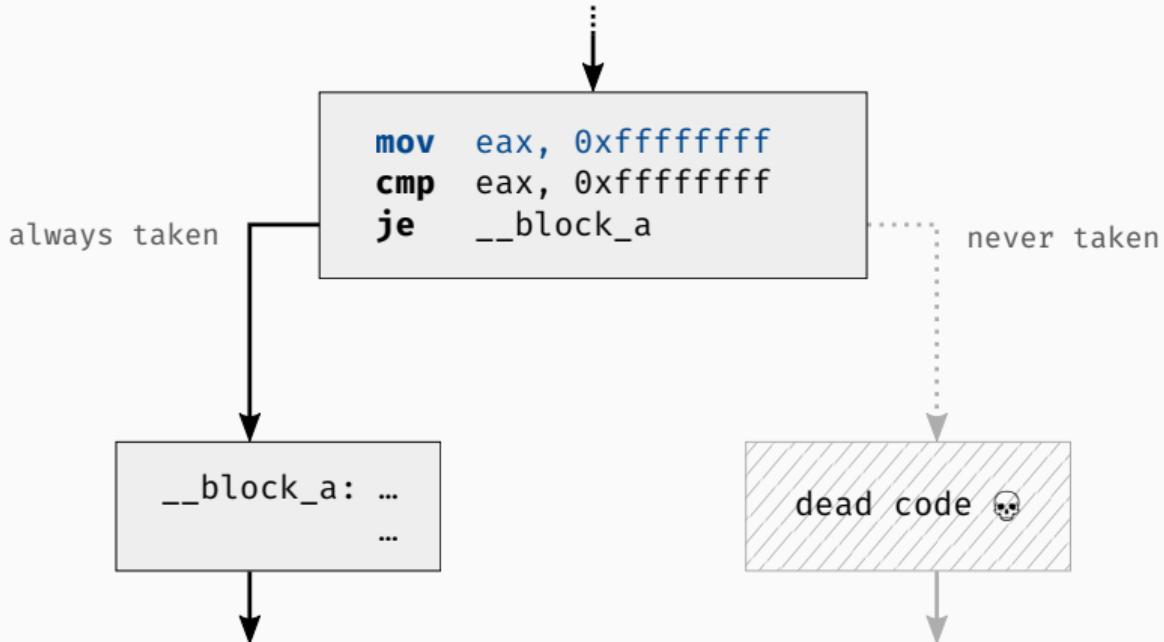
Opaque True Predicate



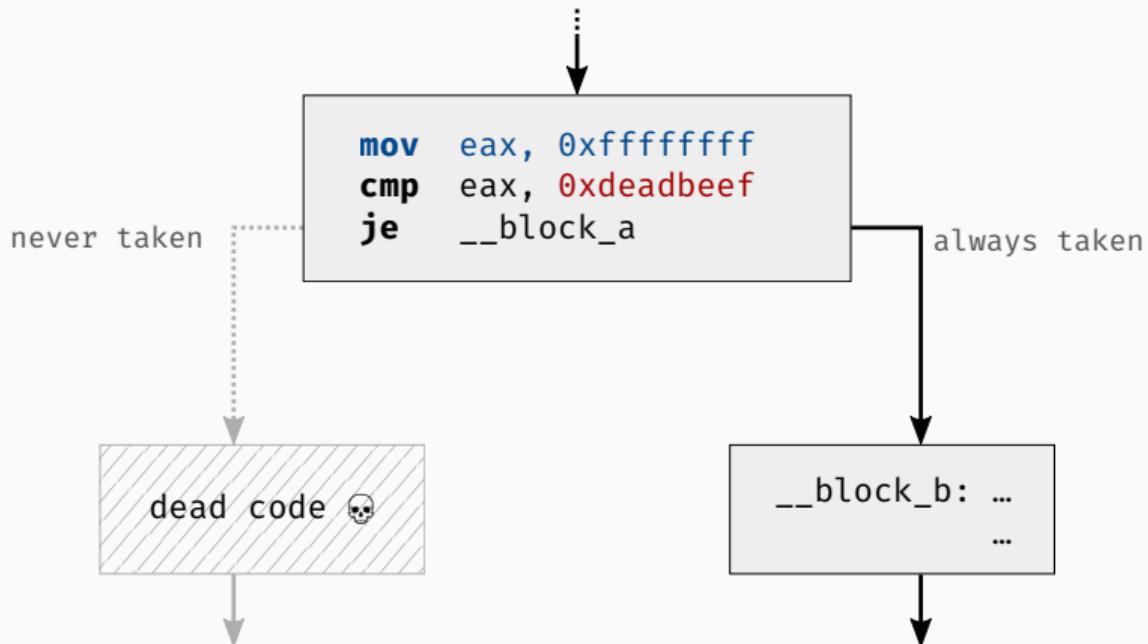
Opaque True Predicate



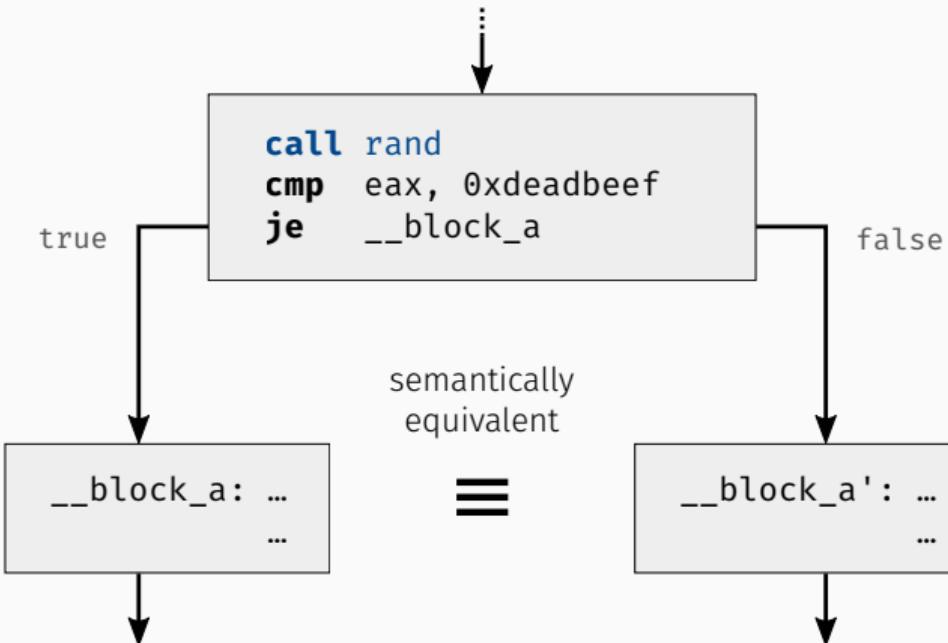
Opaque True Predicate



Opaque True Predicate



Opaque False Predicate



Random Opaque Predicate  
duplicated block

- ⊕ Increase in complexity (branch count, McCabe)
- ⊕ Can be built on hard problems (e.g., aliasing)
- ⊕ Forces analyst to encode additional knowledge
- ⊕ Hard to solve statically

## ⚠ Examples

- `GetCurrentProcess()`  $\Rightarrow -1$
- `fldpi1`  $\Rightarrow st(0) = \pi$
- $x^2 \geq 0 \quad \forall x$
- $x + 1 \neq x \quad \forall x$
- pointer A *must-alias* pointer B
- `checksum(code) = 0x1c43b5cf`

- ⊕ Increase in complexity (branch count, McCabe)
- ⊕ Can be built on hard problems (e.g., aliasing)
- ⊕ Forces analyst to encode additional knowledge
- ⊕ Hard to solve statically
- ⊖ Solved for free using **concrete execution traces**

## ⚠ Examples

- `GetCurrentProcess()`  $\Rightarrow -1$
- `fldpi1`  $\Rightarrow st(0) = \pi$
- $x^2 \geq 0 \quad \forall x$
- $x + 1 \neq x \quad \forall x$
- pointer A *must-alias* pointer B
- `checksum(code) = 0x1c43b5cf`

# Code Obfuscation Techniques

Virtual Machines

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1

__secret_ip:
    mov edx, eax
    add edx, ebx
    mov eax, ebx
    mov ebx, edx
    loop __secret_ip

    mov eax, ebx
    ret
```

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1

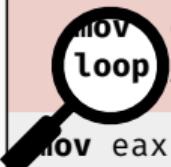
__secret_ip:
    mov edx, eax
    add edx, ebx
    mov eax, ebx
    mov ebx, edx
    loop __secret_ip

    mov eax, ebx
    ret
```

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1

__secret_ip:
    mov edx, eax
    add edx, ebx
    mov eax, ebx
    mov ebx, edx
loop __secret_ip

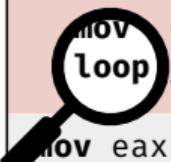
    mov eax, ebx
ret
```



```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1
```

```
--secret_ip:
    mov edx, eax
    add edx, ebx
    mov eax, ebx
    mov ebx, edx
    loop --secret_ip
```

```
    mov eax, ebx
    ret
```



made-up instruction set

```
--bytecode:
    vld r0      vpop r2
    vld r1      vldi #1
    vld r2      vld r3
    vld r1      vsub r3
    vadd r1     vld #0
    vld r2      veq r3
    vpop r0     vbr0 #-0E
```

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1
```

```
--secret_ip:
push __bytecode
call vm_entry
```

```
mov eax, ebx
ret
```



made-up instruction set

```
--bytecode:
db 54 68 69 73 20 64 6f
db 65 73 6e 27 74 20 6c
db 6f 6f 6b 20 6c 69 6b
db 65 20 61 6e 79 74 68
db 69 6e 67 20 74 6f 20
db 6d 65 2e de ad be ef
```

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1
```

```
--secret_ip:
push __bytecode
call vm_entry
```

```
mov eax, ebx
ret
```



made-up instruction set

```
--bytecode:
```

```
db 54 68 69 73 20 64 6f
db 65 73 6e 27 74 20 6c
db 6f 6b 20 6c 69 6b
db 65 20 61 6e 79 74 68
69 6e 67 20 74 6f 20
65 2e de ad be ef
```



## Core Components

**VM Entry/Exit** Context Switch: native context  $\Leftrightarrow$  virtual context

**VM Dispatcher** Fetch–Decode–Execute loop

**Handler Table** Individual VM ISA instruction semantics

- **Entry** Copy native context (registers, flags) to VM context.
- **Exit** Copy VM context back to native context.
- Mapping from native to virtual registers is often 1:1.

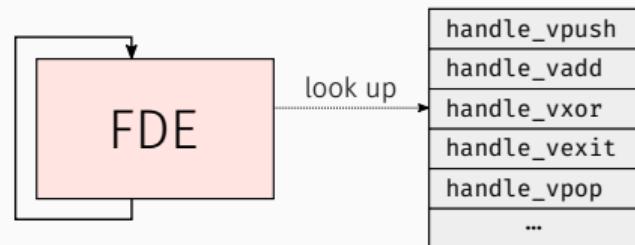
## Core Components

**VM Entry/Exit** Context Switch: native context  $\Leftrightarrow$  virtual context

**VM Dispatcher** Fetch–Decode–Execute loop

**Handler Table** Individual VM ISA instruction semantics

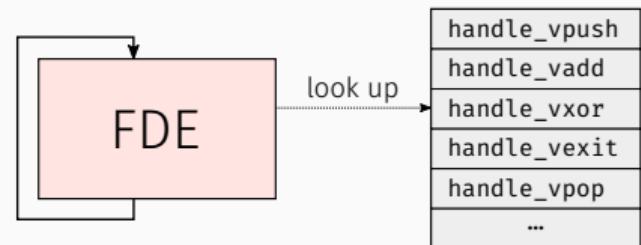
1. Fetch and decode instruction
2. Forward virtual instruction pointer
3. Look up handler for opcode in handler table
4. Invoke handler

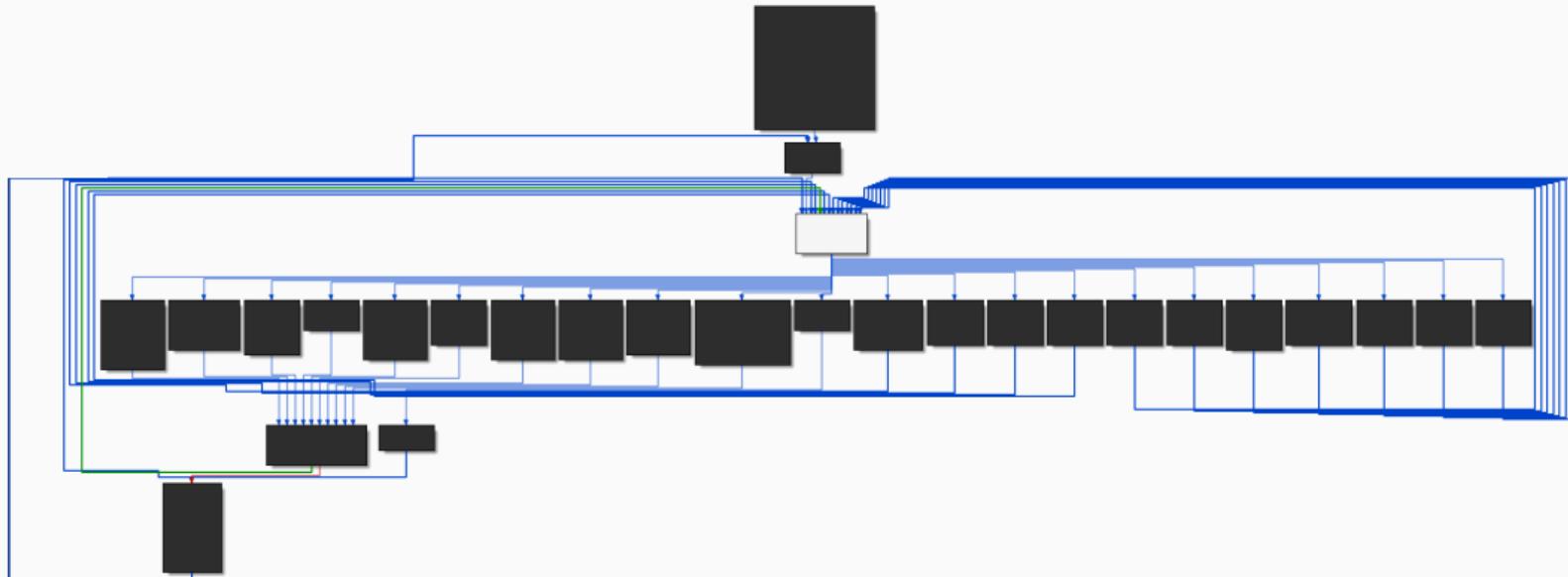


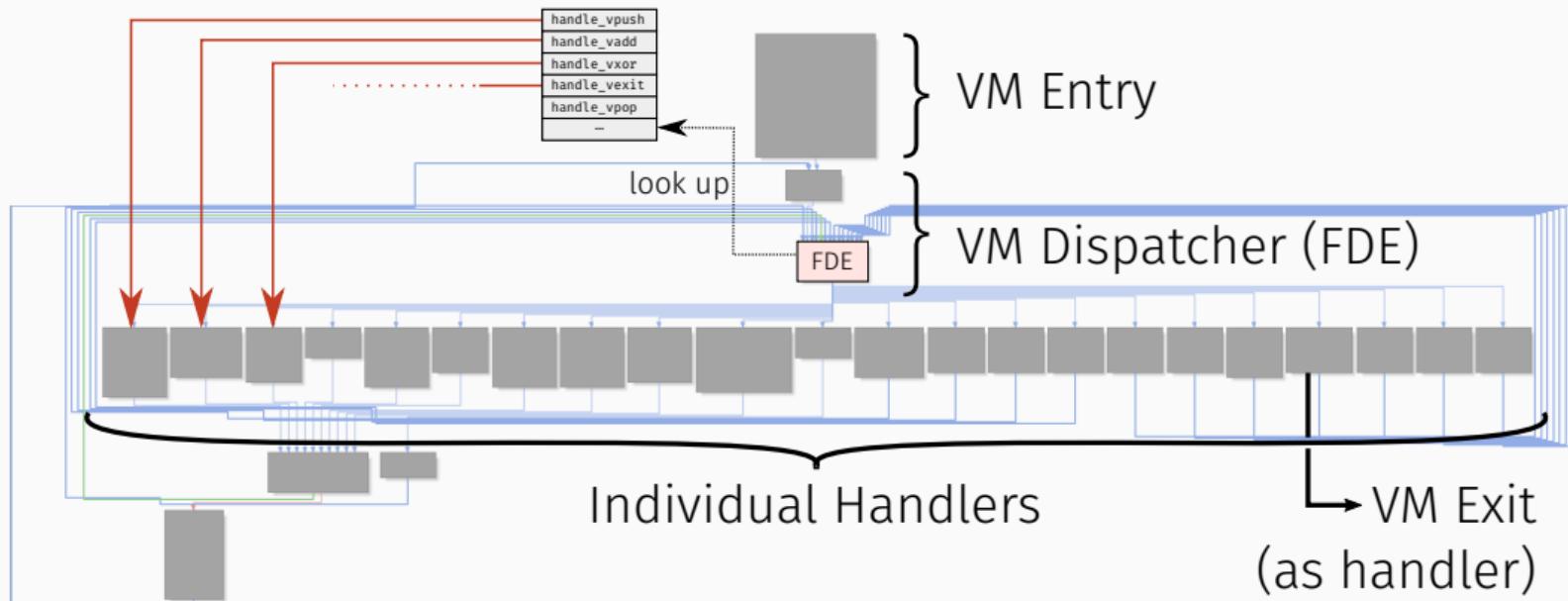
## Core Components

- VM Entry/Exit** Context Switch: native context  $\Leftrightarrow$  virtual context
- VM Dispatcher** Fetch–Decode–Execute loop
- Handler Table** Individual VM ISA instruction semantics

- Table of function pointers indexed by opcode
- One handler per virtual instruction
- Each handler decodes operands and updates VM context







```
__vm_dispatcher:  
    mov    bl, [rsi]  
    inc    rsi  
    movzx  rax, bl  
    jmp    __handler_table[rax * 8]
```

## VM Dispatcher

**rsi** – virtual instruction pointer

**rbp** – VM context

```
__vm_dispatcher:  
    mov    bl, [rsi]  
    inc    rsi  
    movzx  rax, bl  
    jmp    __handler_table[rax * 8]
```

VM Dispatcher

**rsi** – virtual instruction pointer  
**rbp** – VM context

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

Handler performing **nor**  
(with flag side-effects)

# Virtual Machine Hardening

## Hardening Technique #1 – Obfuscating individual VM components.

- Handlers are *conceptually simple*.

## Hardening Technique #1 – Obfuscating individual VM components.

- Handlers are *conceptually simple*.
- Apply traditional code obfuscation transformations:
  - Substitution (`mov rax, rbx` → `push rbx; pop rax`)
  - Opaque Predicates
  - Junk Code
  - ...

```
mov eax, dword [rbp]
mov ecx, dword [rbp+4]
cmp r11w, r13w
sub rbp, 4
not eax
clc
cmc
cmp rdx, 0x28b105fa
not ecx
cmp r12b, r9b
```

## Hardening Technique #2 – Duplicating VM handlers.

- Handler table is typically indexed using one byte (= 256 entries).

## Hardening Technique #2 – Duplicating VM handlers.

- Handler table is typically indexed using one byte (= 256 entries).
- **Idea:** *Duplicate* existing handlers to populate full table.
- Use traditional obfuscation techniques to impede *code similarity* analyses.

**Goal:** Increase workload of reverse engineer.

handle\_vpush

handle\_vadd

handle\_vnor

handle\_vpop

handle_vpush
handle_vadd
handle_vnor
handle_vpop



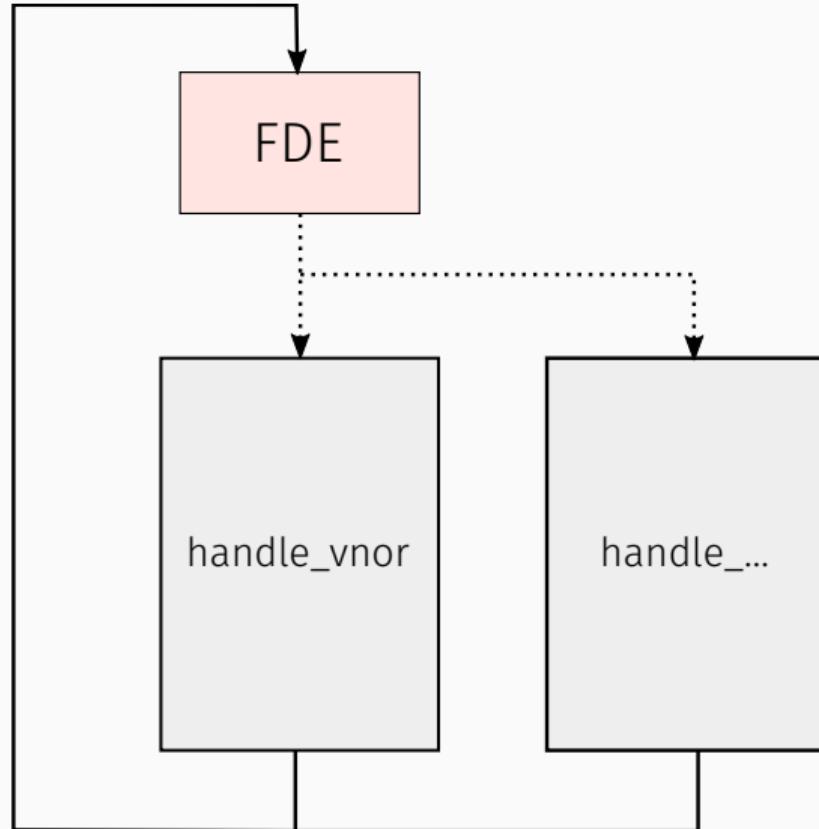
handle_vpush
handle_vadd
handle_vnor ''
handle_vpop
handle_vadd'
handle_vnor
handle_vnor '
handle_vadd ''

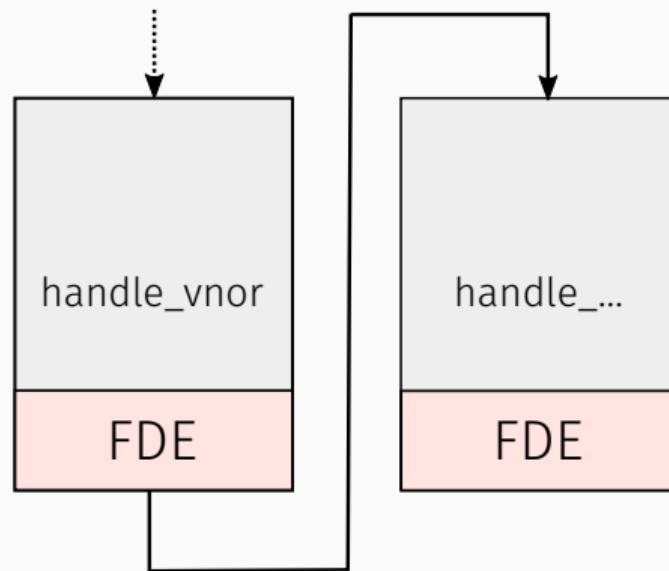
## Hardening Technique #3 – No central VM dispatcher.

- A *central* VM dispatcher allows attacker to easily observe VM execution.
- **Idea:** Instead of branching to the central dispatcher, *inline* it into each handler.

**Goal:** No “single point of failure”.

(Themida, VMProtect Demo)





# Threaded Code

James R. Bell  
Digital Equipment Corporation

The concept of "threaded code" is presented as an alternative to machine language code. Hardware and software realizations of it are given. In software it is realized as interpretive code not needing an interpreter. Extensions and optimizations are mentioned.

**Key Words and Phrases:** interpreter, machine code, time tradeoff, space tradeoff, compiled code, subroutine calls, threaded code

**CR Categories:** 4.12, 4.13, 6.33

Fig. 2 Flow of control: interpretive code.

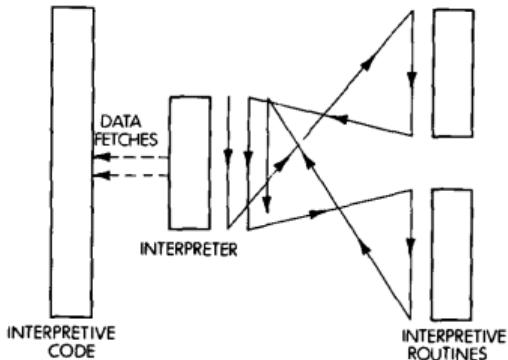
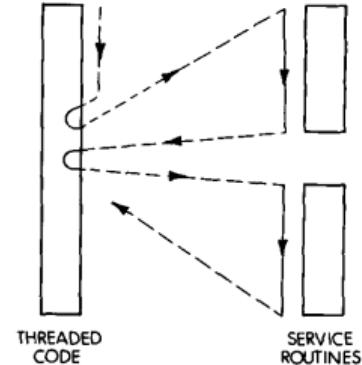


Fig. 3. Flow of control: threaded code.



## Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.

## Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.
- **Idea:** Instead of querying an explicit handler table,  
*encode* the next handler address in the VM instruction itself.

**Goal:** Hide location of handlers that have not been executed yet.

(VMProtect Full, SolidShield)

## Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.
- Idea:  Handler table,  
the VM instruction itself.

**Goal:** Hide location of handlers that have not been executed yet.

(VMProtect Full, SolidShield)

## Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.



**Goal:** Hide location of handlers that have not been executed yet.

(VMProtect Full, SolidShield)

SOFTWARE-PRACTICE AND EXPERIENCE, VOL. 11, 963-973 (1981)

# Interpretation Techniques<sup>\*</sup>

PAUL KLINT

*Mathematical Centre, P.O. Box 4079, 1009AB Amsterdam, The Netherlands*

## SUMMARY

The relative merits of implementing high level programming languages by means of interpretation or compilation are discussed. The properties and the applicability of interpretation techniques known as classical interpretation, direct threaded code and indirect threaded code are described and compared.

**KEY WORDS**      Interpretation versus compilation   Interpretation techniques   Instruction encoding   Code generation   Direct threaded code   Indirect threaded code.

## Hardening Technique #5 – Blinding VM bytecode.

- *Global analyses* on the bytecode possible, easy to patch instructions.

## Hardening Technique #5 – Blinding VM bytecode.

- *Global analyses* on the bytecode possible, easy to patch instructions.
- Idea:
  - *Flow-sensitive* instruction decoding (“decryption” based on key register).
  - Custom decryption routine per handler, diversification.
  - Patching requires re-encryption of subsequent bytecode.

**Goal:** Hinder global analyses of bytecode and patching.

*operand*                     $\leftarrow [vIP + 0]$

*context*                     $\leftarrow \text{semantics}(\text{context}, \text{operand})$   
*next\_handler*             $\leftarrow [vIP + 4]$

$vIP \leftarrow vIP + 8$

**jmp** *next\_handler*

*operand*

$\leftarrow [vIP + 0]$

 *operand*

$\leftarrow \text{unmangle}(\textit{operand}, \textbf{key})$

 **key**

$\leftarrow \text{unmangle}'(\textbf{key}, \textit{operand})$

*context*

$\leftarrow \text{semantics}(\textit{context}, \textit{operand})$

*next\_handler*

$\leftarrow [vIP + 4]$

 *next\_handler*

$\leftarrow \text{unmangle}''(\textit{next\_handler}, \textbf{key})$

 **key**

$\leftarrow \text{unmangle}'''(\textbf{key}, \textit{next\_handler})$

$vIP \leftarrow vIP + 8$

**jmp** *next\_handler*

# Code Obfuscation Techniques

Mixed Boolean-Arithmetic

What does this expression compute?

$$(x \oplus y) + 2 \cdot (x \wedge y)$$

What does this expression compute?

$$\begin{aligned}(x \oplus y) + 2 \cdot (x \wedge y) \\ = x + y\end{aligned}$$

What does this expression compute?

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

What does this expression compute?

$$\begin{aligned} & (((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z) \\ &= x + y + z \end{aligned}$$

- Boolean identities?
- Arithmetic identities?
- Karnaugh-Veitch maps?

$$A \cdot 0 = 0$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

$$x^2 - y^2 = (x + y)(x - y)$$

		AB		
		00	01	11
θ	00	0	0	1
	01	0	0	1
	11	0	0	1
	10	0	1	1
	01	1	1	1

## Boolean-arithmetic algebra BA[n]

$(B^n, \wedge, \vee, \oplus, \neg, \leq, \geq, >, <, \leq^s, \geq^s, >^s, <^s, \neq, =, \gg^s, \gg, \ll, +, -, \cdot)$   
is a Boolean-arithmetic algebra BA[n], for  $n > 0$ ,  $B = \{0, 1\}$ .

BA[n] includes, amongst others, both:

- Boolean algebra  $(B^n, \wedge, \vee, \neg)$ ,
- Integer modular ring  $\mathbb{Z}/(2^n)$ .

No techniques to simplify such expressions easily!

# Deobfuscation

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:
```

- **mov** rcx, [rbp]
- mov** rbx, [rbp + 4]
- not** rcx
- not** rbx
- and** rcx, rbx
- mov** [rbp + 4], rcx
- pushf**
- pop** [rbp]
- jmp** \_\_vm\_dispatcher

rcx  $\leftarrow$  [rbp]

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
• mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

rcx  $\leftarrow$  [rbp]  
rbx  $\leftarrow$  [rbp + 4]

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    • not   rcx  
    not   rbx  
    and   rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg$ rcx =  $\neg$ [rbp]
```

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
• not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg \text{rcx} = \neg [\text{rbp}]$   
rbx ←  $\neg \text{rbx} = \neg [\text{rbp} + 4]$ 
```

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    • and   rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

$$\begin{aligned} \text{rcx} &\leftarrow [\text{rbp}] \\ \text{rbx} &\leftarrow [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \neg \text{rcx} = \neg [\text{rbp}] \\ \text{rbx} &\leftarrow \neg \text{rbx} = \neg [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \text{rcx} \wedge \text{rbx} \\ &= (\neg [\text{rbp}]) \wedge (\neg [\text{rbp} + 4]) \end{aligned}$$

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    • and   rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

$$\begin{aligned} \text{rcx} &\leftarrow [\text{rbp}] \\ \text{rbx} &\leftarrow [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \neg \text{rcx} = \neg [\text{rbp}] \\ \text{rbx} &\leftarrow \neg \text{rbx} = \neg [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \text{rcx} \wedge \text{rbx} \\ &= (\neg [\text{rbp}]) \wedge (\neg [\text{rbp} + 4]) \\ &= [\text{rbp}] \downarrow [\text{rbp} + 4] \end{aligned}$$

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov rcx, [rbp]  
    mov rbx, [rbp + 4]  
    not rcx  
    not rbx  
    and rcx, rbx  
• mov [rbp + 4], rcx  
  pushf  
  pop [rbp]  
  jmp __vm_dispatcher
```

$$\begin{aligned} \text{rcx} &\leftarrow [\text{rbp}] \\ \text{rbx} &\leftarrow [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \neg \text{rcx} = \neg [\text{rbp}] \\ \text{rbx} &\leftarrow \neg \text{rbx} = \neg [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \text{rcx} \wedge \text{rbx} \\ &= (\neg [\text{rbp}]) \wedge (\neg [\text{rbp} + 4]) \\ &= [\text{rbp}] \downarrow [\text{rbp} + 4] \\ [\text{rbp} + 4] &\leftarrow \text{rcx} = [\text{rbp}] \downarrow [\text{rbp} + 4] \end{aligned}$$

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov rcx, [rbp]  
    mov rbx, [rbp + 4]  
    not rcx  
    not rbx  
    and rcx, rbx  
    mov [rbp + 4], rcx  
• pushf  
    pop [rbp]  
    jmp __vm_dispatcher
```

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg \text{rcx} = \neg [\text{rbp}]$   
rbx ←  $\neg \text{rbx} = \neg [\text{rbp} + 4]$   
rcx ←  $\text{rcx} \wedge \text{rbx}$   
=  $(\neg [\text{rbp}] \wedge \neg [\text{rbp} + 4])$   
=  $[\text{rbp}] \downarrow [\text{rbp} + 4]$   
[rbp + 4] ←  $\text{rcx} = [\text{rbp}] \downarrow [\text{rbp} + 4]$   
  
rsp ←  $\text{rsp} - 4$   
[rsp] ← flags
```

Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov rcx, [rbp]  
    mov rbx, [rbp + 4]  
    not rcx  
    not rbx  
    and rcx, rbx  
    mov [rbp + 4], rcx  
    pushf  
• pop [rbp]  
jmp __vm_dispatcher
```

Handler performing **nor**  
(with flag side-effects)

$$\begin{aligned} \text{rcx} &\leftarrow [\text{rbp}] \\ \text{rbx} &\leftarrow [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \neg \text{rcx} = \neg [\text{rbp}] \\ \text{rbx} &\leftarrow \neg \text{rbx} = \neg [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \text{rcx} \wedge \text{rbx} \\ &= (\neg [\text{rbp}]) \wedge (\neg [\text{rbp} + 4]) \\ &= [\text{rbp}] \downarrow [\text{rbp} + 4] \\ [\text{rbp} + 4] &\leftarrow \text{rcx} = [\text{rbp}] \downarrow [\text{rbp} + 4] \\ \\ \text{rsp} &\leftarrow \text{rsp} - 4 \\ [\text{rsp}] &\leftarrow \text{flags} \\ [\text{rbp}] &\leftarrow [\text{rsp}] = \text{flags} \\ \text{rsp} &\leftarrow \text{rsp} + 4 \end{aligned}$$

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    and  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
• jmp   __vm_dispatcher
```

Handler performing **nor**  
(with flag side-effects)

$$\begin{aligned} \text{rcx} &\leftarrow [\text{rbp}] \\ \text{rbx} &\leftarrow [\text{rbp} + 4] \\ \text{rcx} &\leftarrow \neg \text{rcx} = \neg [\text{rbp}] \\ \text{rbx} &\leftarrow \neg \text{rbx} = \neg [\text{rbp} + 4] \end{aligned}$$
$$[\text{rbp} + 4] \leftarrow ([\text{rbp}] \downarrow [\text{rbp} + 4])$$
$$= [\text{rbp}] \downarrow [\text{rbp} + 4]$$
$$[\text{rbp} + 4] \leftarrow \text{rcx} = [\text{rbp}] \downarrow [\text{rbp} + 4]$$
$$\text{rsp} \leftarrow \text{rsp} - 4$$
$$[\text{rsp}] \leftarrow \text{flags}$$
$$[\text{rbp}] \leftarrow [\text{rsp}] = \text{flags}$$
$$\text{rsp} \leftarrow \text{rsp} + 4$$

# Virtual Machine Handler

<b>mov</b>	eax, dword [rbp]	<b>jmp</b>	0xfffffffffffff63380
<b>mov</b>	ecx, dword [rbp + 4]	<b>dec</b>	eax
<b>cmp</b>	r11w, r13w	<b>stc</b>	
<b>sub</b>	rbp, 4	<b>ror</b>	eax, 1
<b>not</b>	eax	<b>jmp</b>	0xfffffffffffff2a70
<b>clc</b>		<b>dec</b>	eax
<b>cmc</b>		<b>clc</b>	
<b>cmp</b>	rdx, 0x28b105fa	<b>bswap</b>	eax
<b>not</b>	ecx	<b>test</b>	bp, 0x5124
<b>cmp</b>	r12b, r9b	<b>neg</b>	eax
<b>cmc</b>		<b>test</b>	dil, 0xe9
<b>and</b>	eax, ecx	<b>cmp</b>	bx, r14w
<b>jmp</b>	0xc239	<b>cmc</b>	
<b>mov</b>	word [rbp + 8], eax	<b>push</b>	rbx
<b>pushfq</b>		<b>sub</b>	bx, 0x49f8
<b>movzx</b>	eax, r10w	<b>xor</b>	dword [rsp], eax
<b>and</b>	ax, di	<b>and</b>	bh, 0xaf
<b>pop</b>	qword [rbp]	<b>pop</b>	rbx
<b>sub</b>	rsi, 4	<b>movsx</b>	rax, eax
<b>shld</b>	rax, rdx, 0x1b	<b>test</b>	r13b, 0x94
<b>xor</b>	ah, 0x4d	<b>add</b>	rdi, rax
<b>mov</b>	eax, dword [rsi]	<b>jmp</b>	0xfffffffffffffc67c7
<b>cmp</b>	ecx, r11d	<b>lea</b>	rax, [rsp + 0x140]
<b>test</b>	r10, 0x179708d5	<b>cmp</b>	rbp, rax
<b>xor</b>	eax, ebx	<b>ja</b>	0x6557b
		<b>jmp</b>	rdi

# Virtual Machine Handler

# Virtual Machine Handler

**M<sub>1</sub>** = (¬M<sub>1</sub>) ∧ (¬M<sub>2</sub>)

# Mixed Boolean-Arithmetic Expression

```
int mixed_boolean(int A, int B, int C) {
    int result;

    result = (((1438524315 + (((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) |
        (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265)) + 1438524315 * ((2956783114 -
        -1478456685 * (((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) | (-1478456685 *
            (1668620215 - A) - 2956783115))) + A) - 1553572265)) | (-1478456685 * (1668620215 - B) -
        2956783115)) - ((1438524315 + (1668620215 - (((1438524315 + C) + 1438524315 * ((2956783114 -
            -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) - 1553572265)) +
        1438524315 * ((2956783114 - -1478456685 * (1668620215 - (((1438524315 + C) + 1438524315 *
            (2956783114 - -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115))) + A) -
            1553572265)) | (-1478456685 * B - 2956783115))) + 1553572265;

    return -1478456685 * result - 2956783115;
}
```

# Mixed Boolean-Arithmetic Expression

# Mixed Boolean-Arithmetic Expression

- ⊕ Captures full semantics of executed code
- ⊕ Computer algebra system, some degree of simplification
- ⊖ Usability decreases with increasing *syntactic* complexity
  - Artificial complexity (substitution, ...)
  - Algebraic complexity (MBA)

- ⊕ Captures full semantics of executed code
- ⊕ Computer algebra system, some degree of simplification
- ⊖ Usability decreases with increasing syntactic complexity
  - Artificial complexity (substitution, ...)
  - Algebraic complexity (MBA)

What if we could reason about *semantics* only instead of *syntax*?

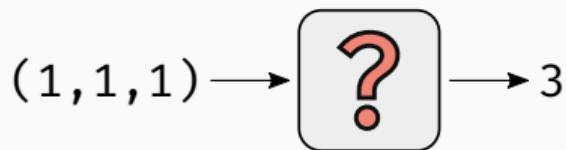
# Program Synthesis

We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

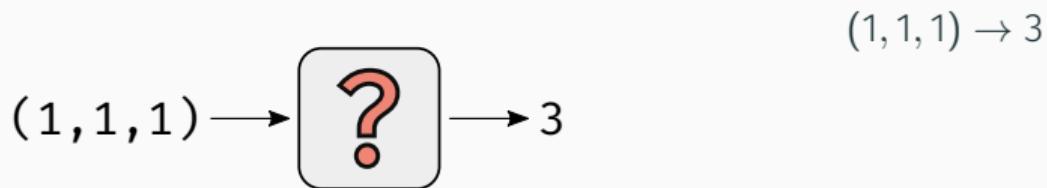
We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



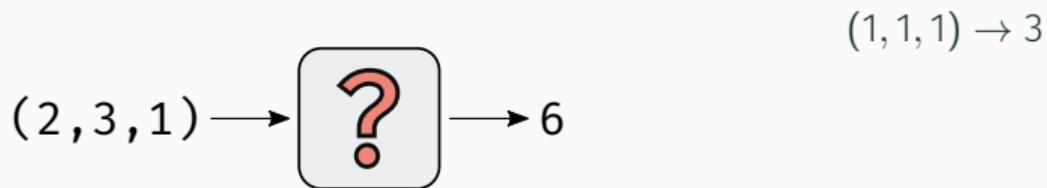
We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



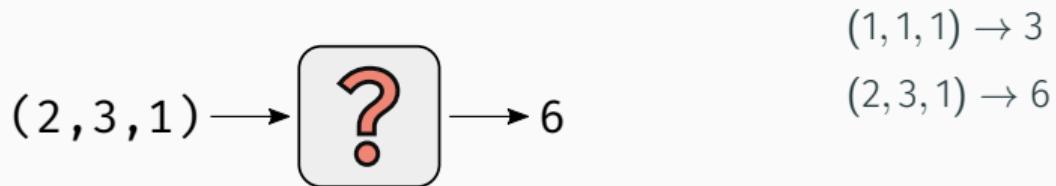
We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



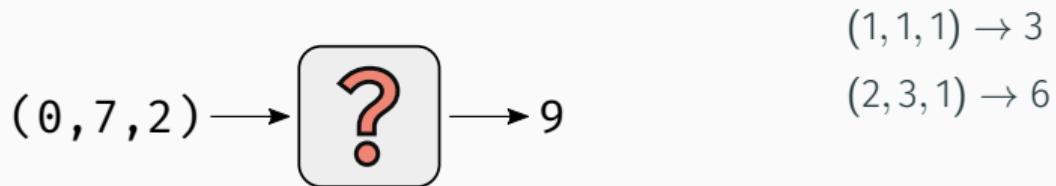
We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



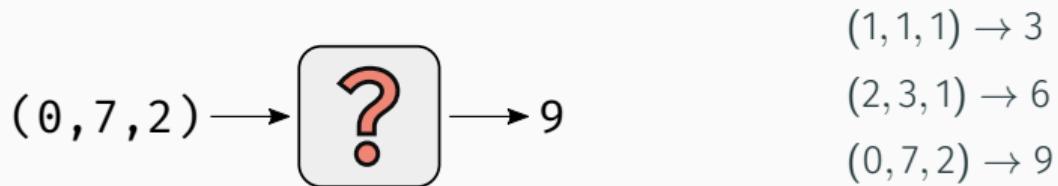
We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

$$(1, 1, 1) \rightarrow 3$$

$$(2, 3, 1) \rightarrow 6$$

$$(0, 7, 2) \rightarrow 9$$

We **learn** a function that has the same I/O behavior:

We use  $f$  as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

$$(1, 1, 1) \rightarrow 3$$

$$(2, 3, 1) \rightarrow 6$$

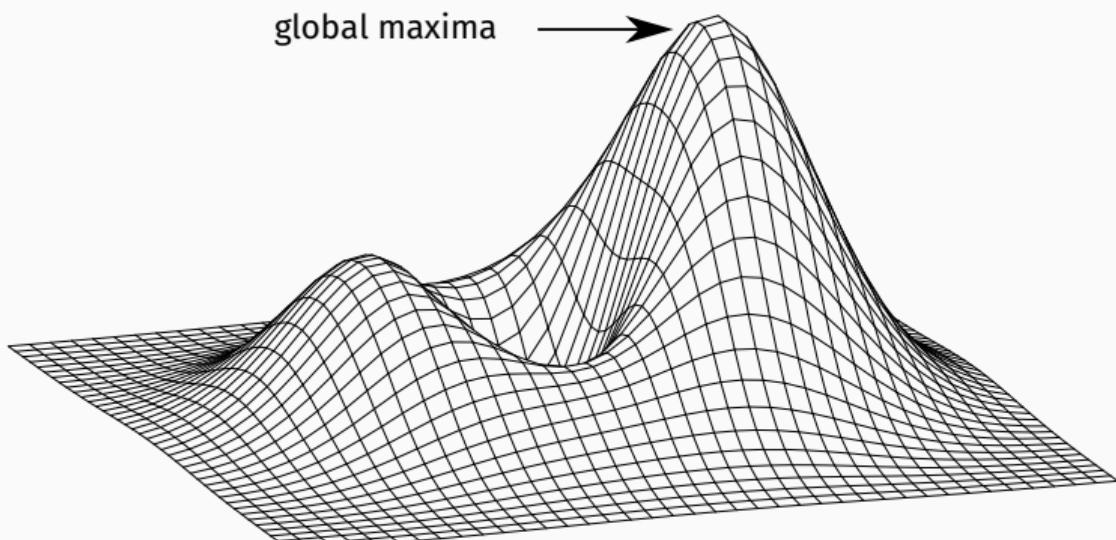
$$(0, 7, 2) \rightarrow 9$$

We **learn** a function that has the same I/O behavior:

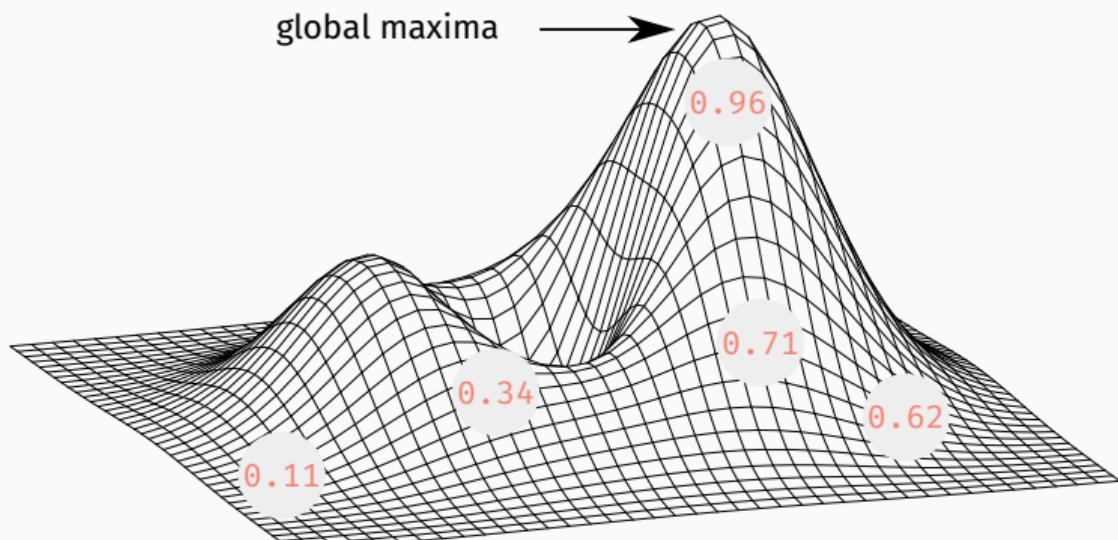
$$h(x, y, z) := x + y + z$$

How to synthesize programs?

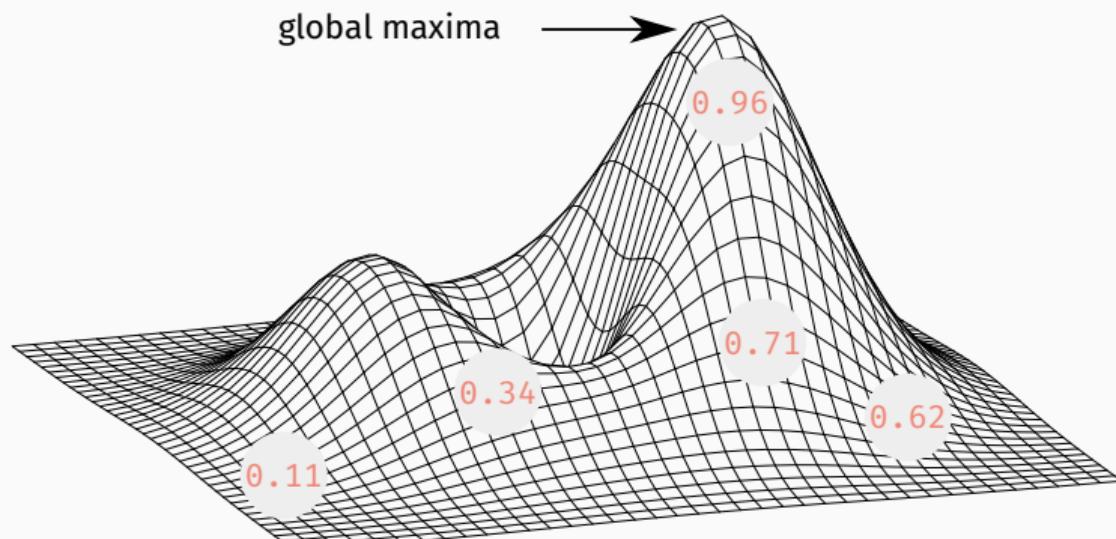
- probabilistic optimization problem



- probabilistic optimization problem



- probabilistic optimization problem
- based on Monte Carlo Tree Search (MCTS)



Let's synthesize:  $a + b \bmod 8$

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

- non-terminal symbol:  $U$

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

- non-terminal symbol:  $U$
- input variables:  $\{a, b\}$

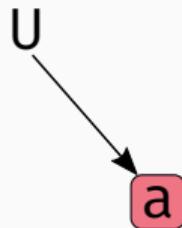
$$U \rightarrow U + U \mid U * U \mid a \mid b$$

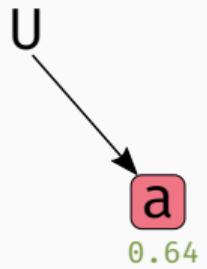
- non-terminal symbol:  $U$
- input variables:  $\{a, b\}$
- candidate programs:  $a, b, a * b, a + b, \dots$

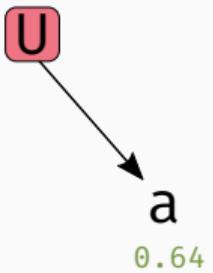
$$U \rightarrow U + U \mid U * U \mid a \mid b$$

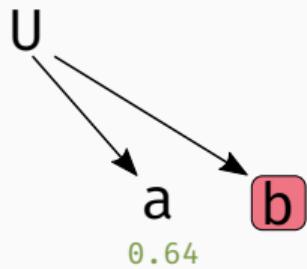
- non-terminal symbol:  $U$
- input variables:  $\{a, b\}$
- candidate programs:  $a, b, a * b, a + b, \dots$
- intermediate programs:  $U + U, U * U, U + b, \dots$

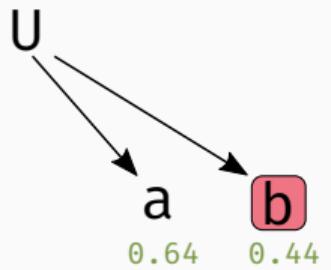


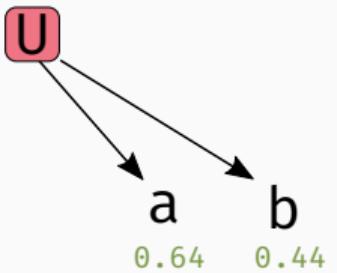


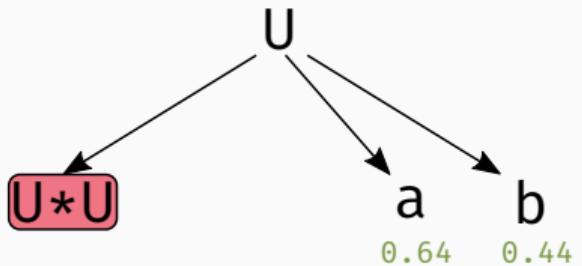


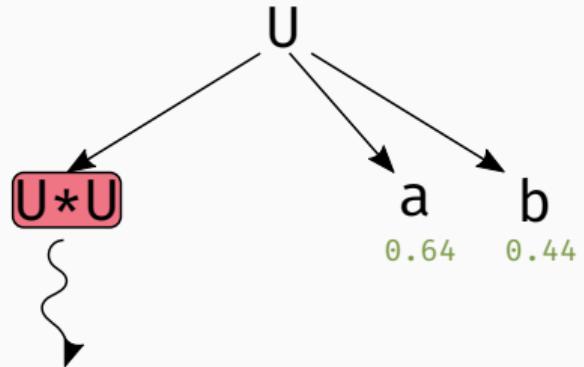


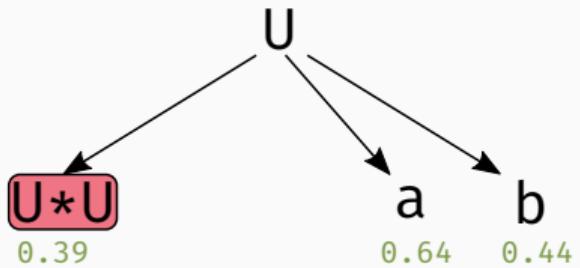


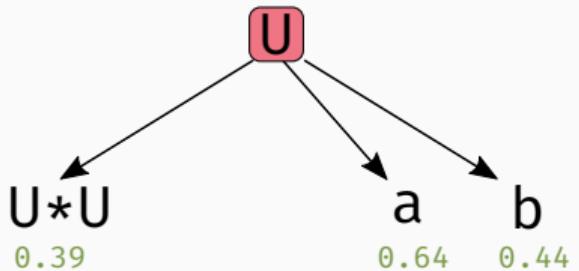


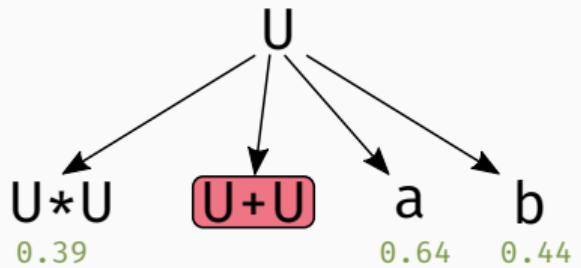


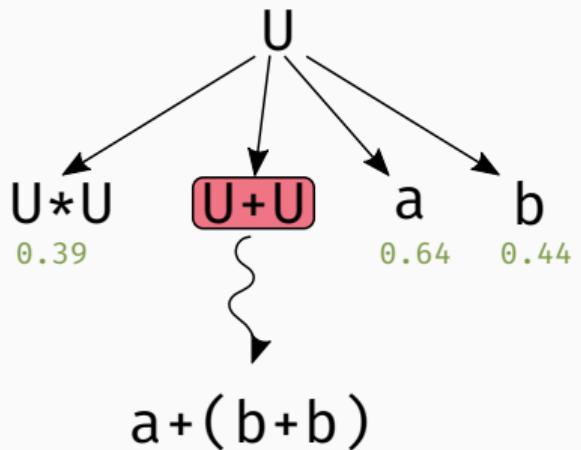


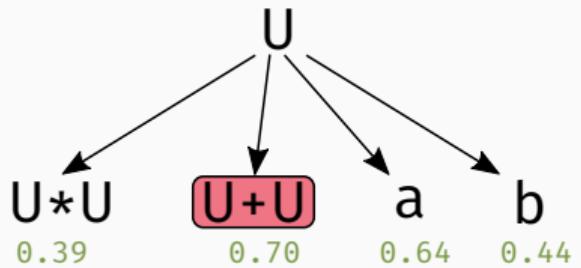

$$(a+a)*(b*a)$$

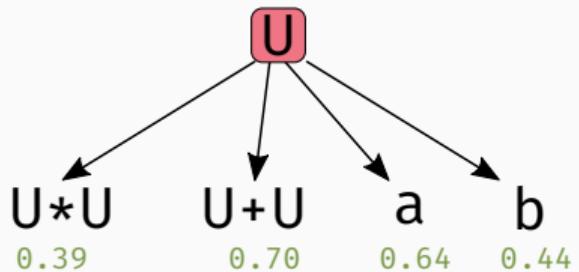


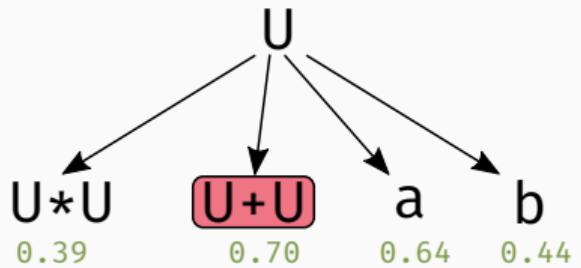


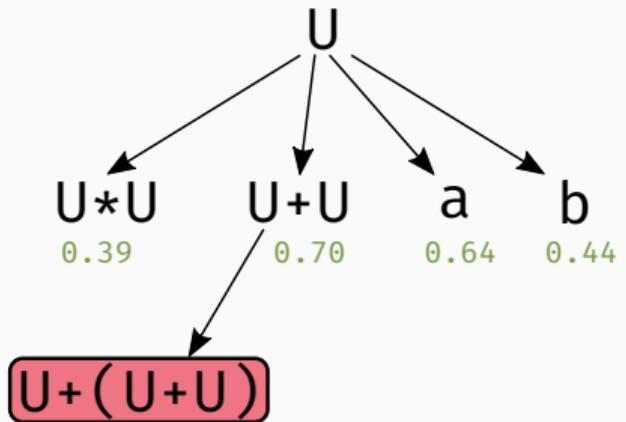


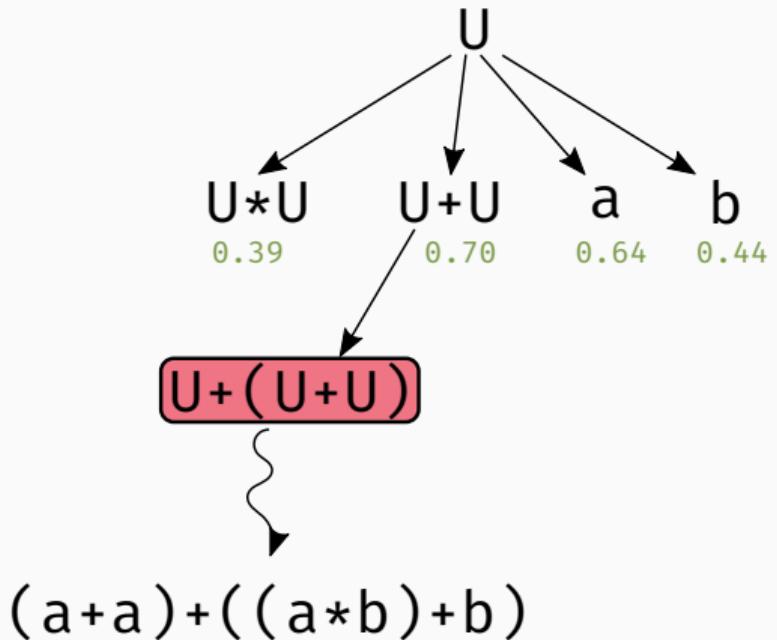


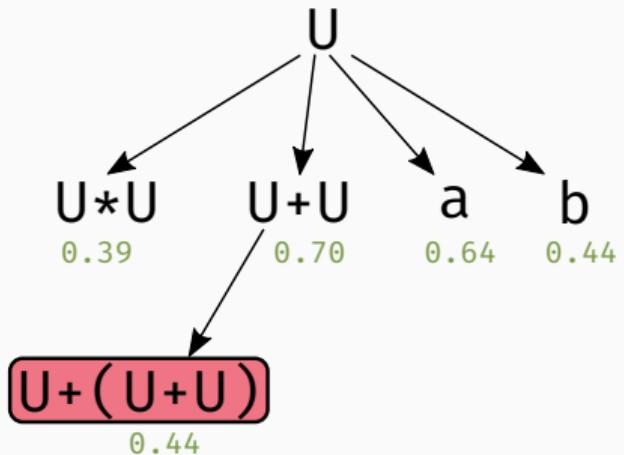


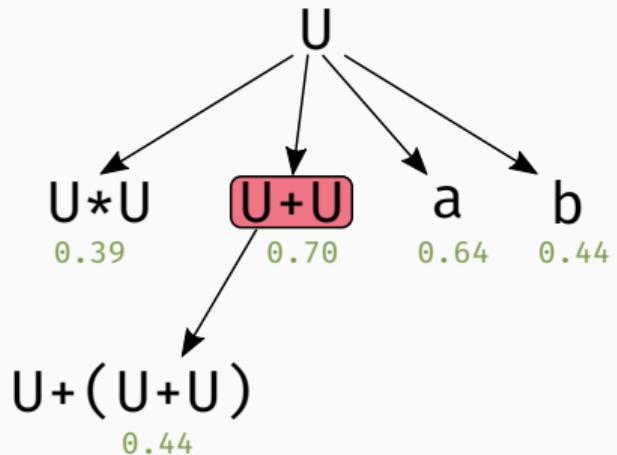


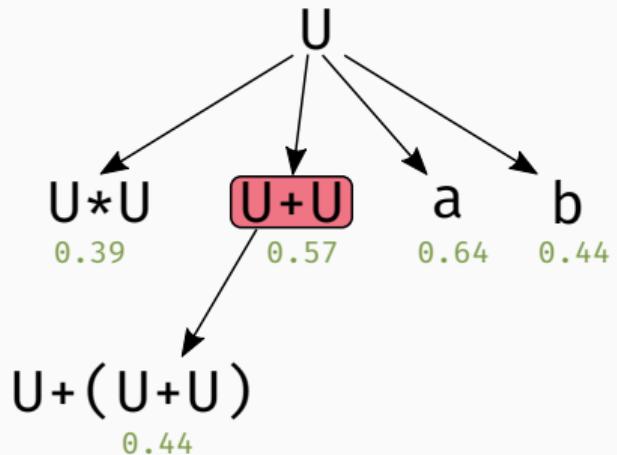


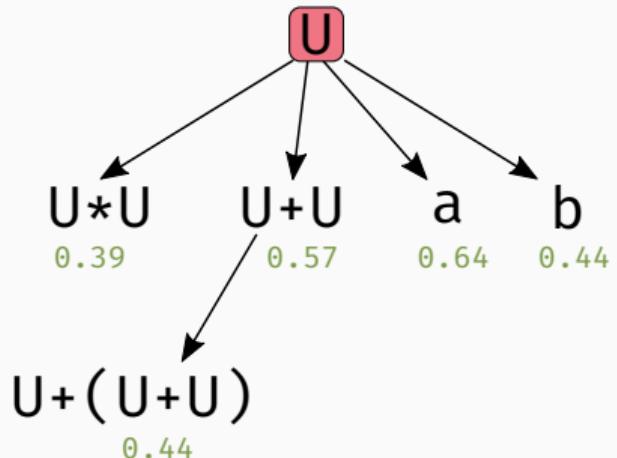


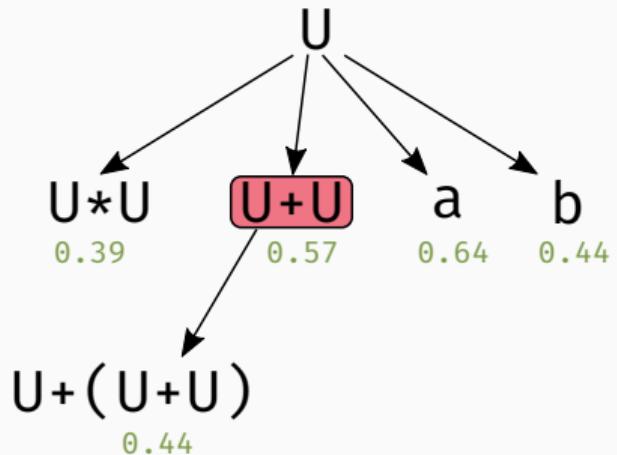


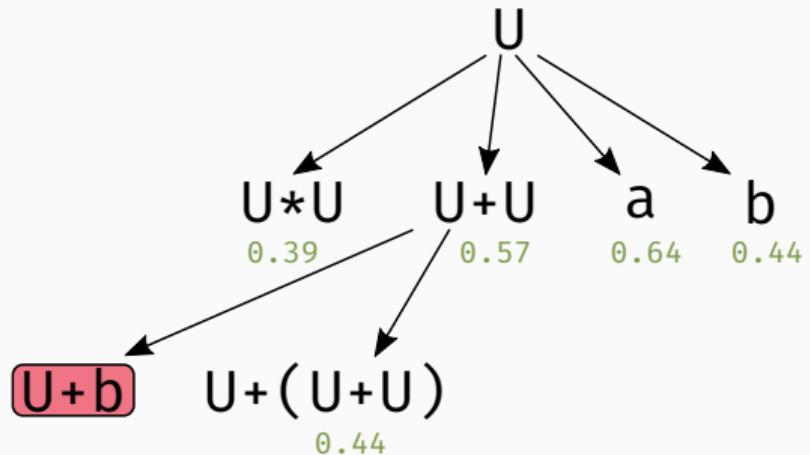


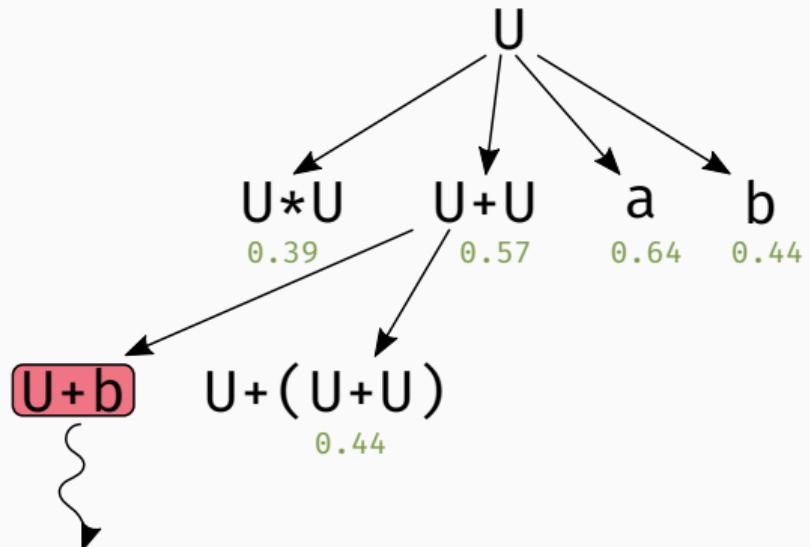


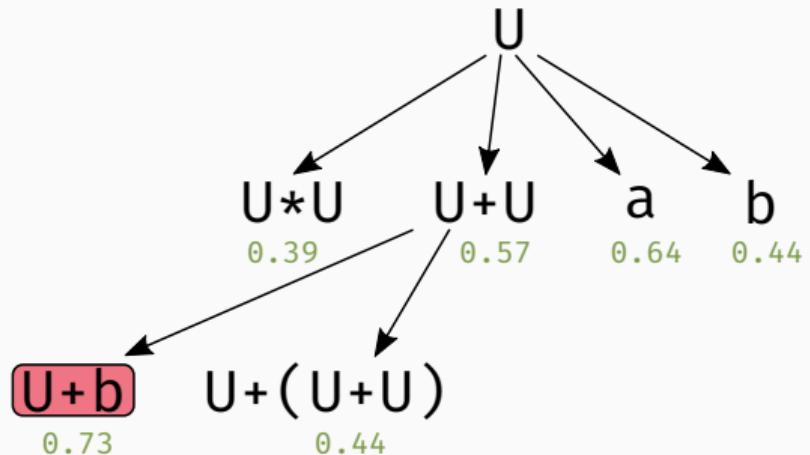


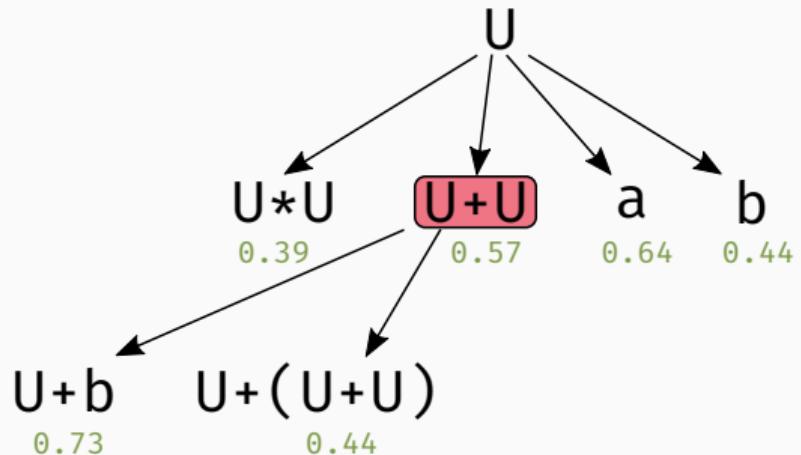


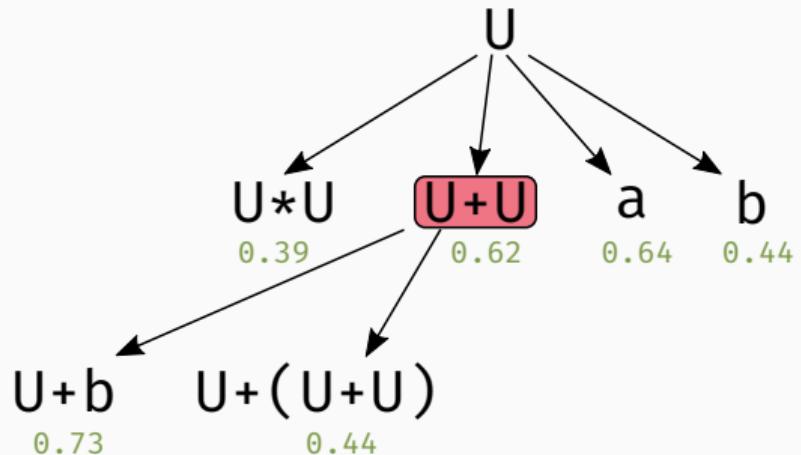


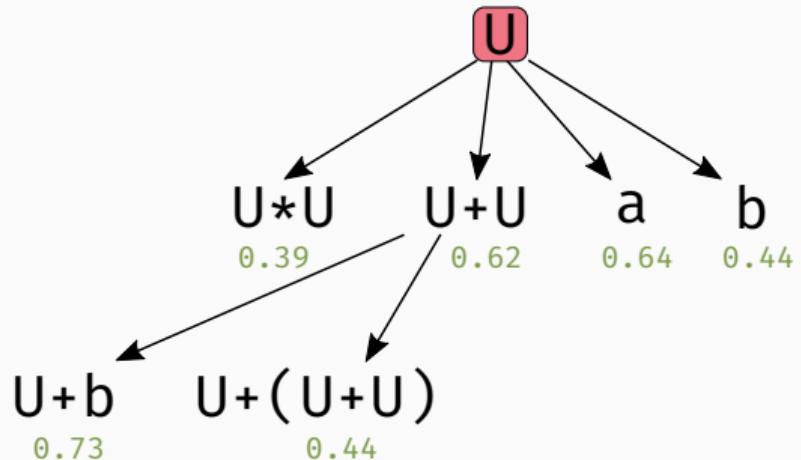


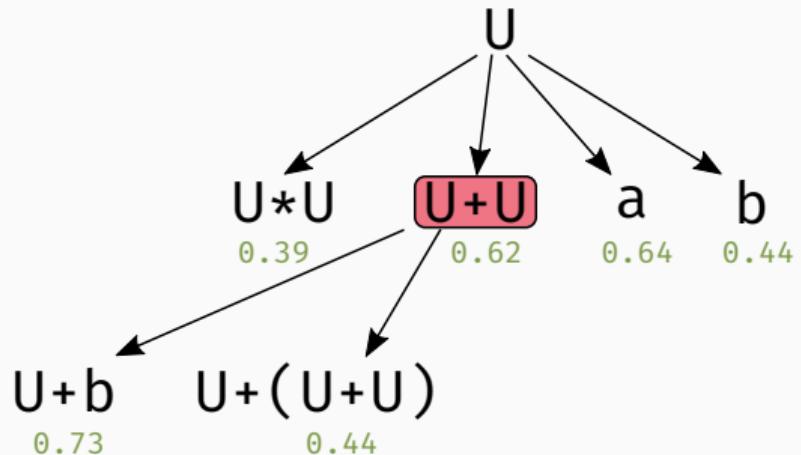


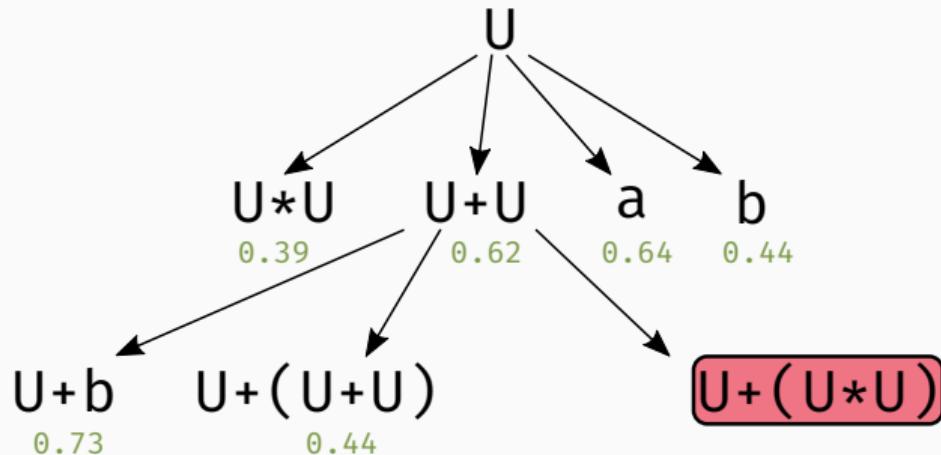


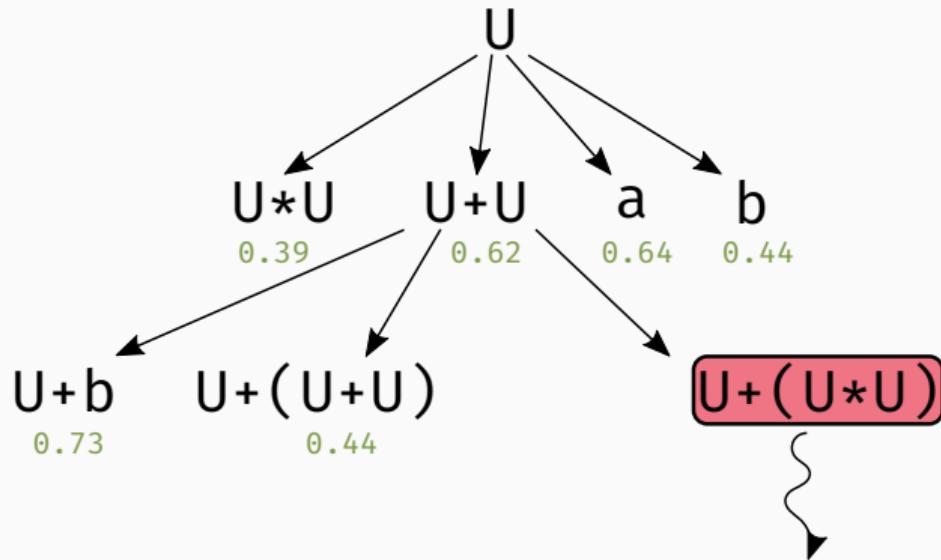


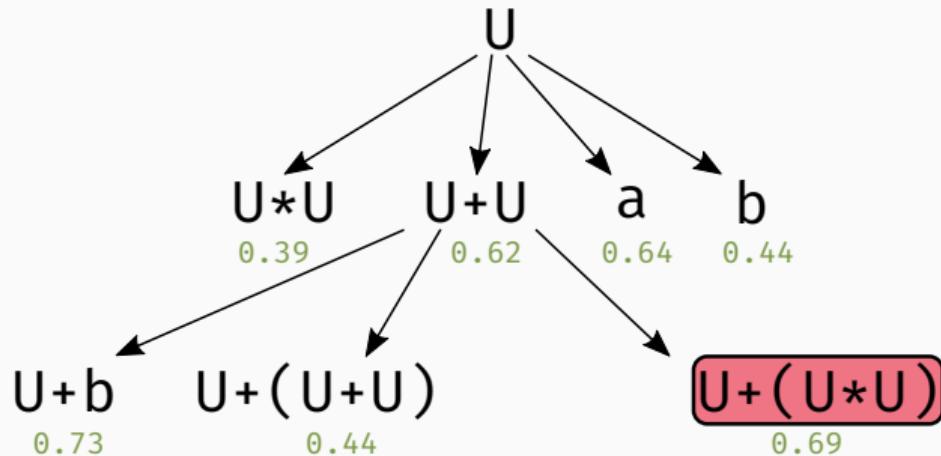


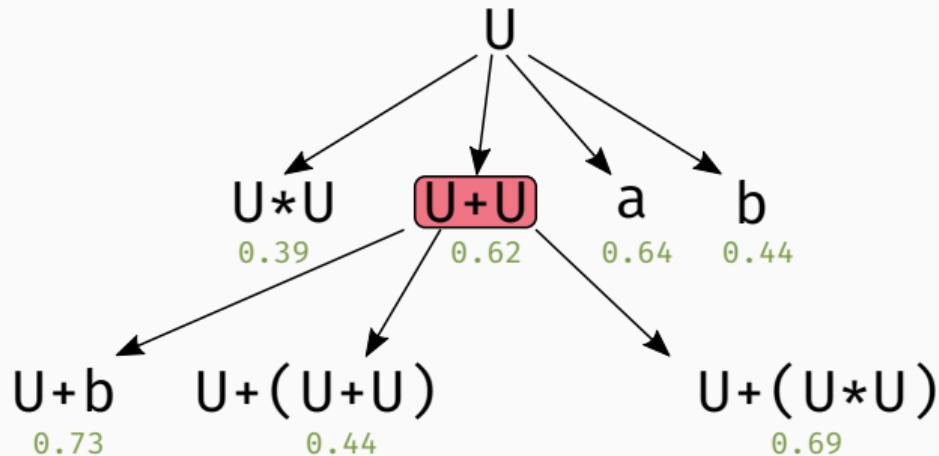


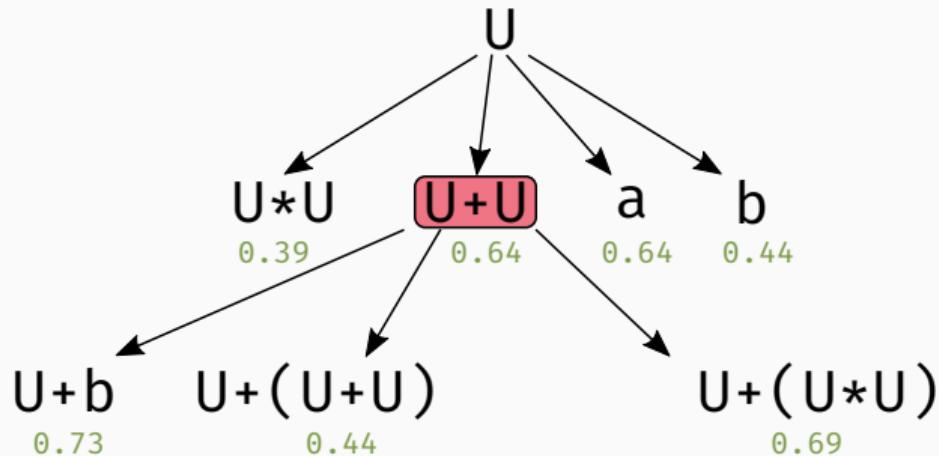


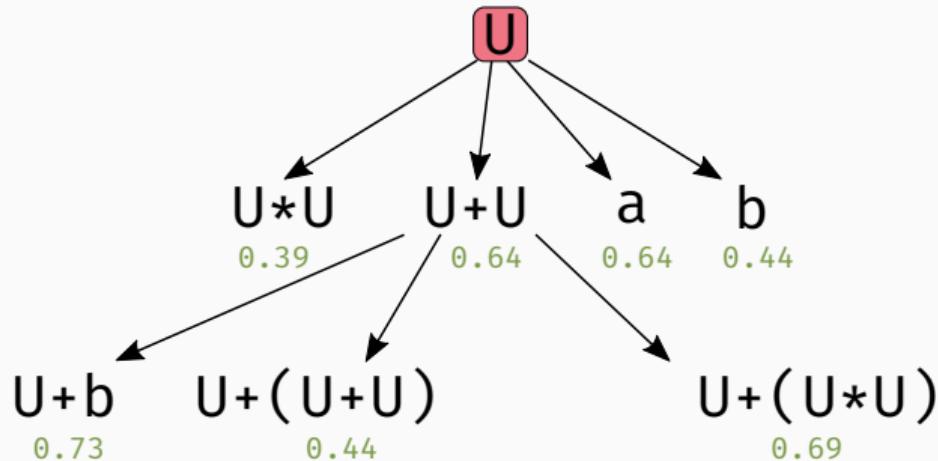


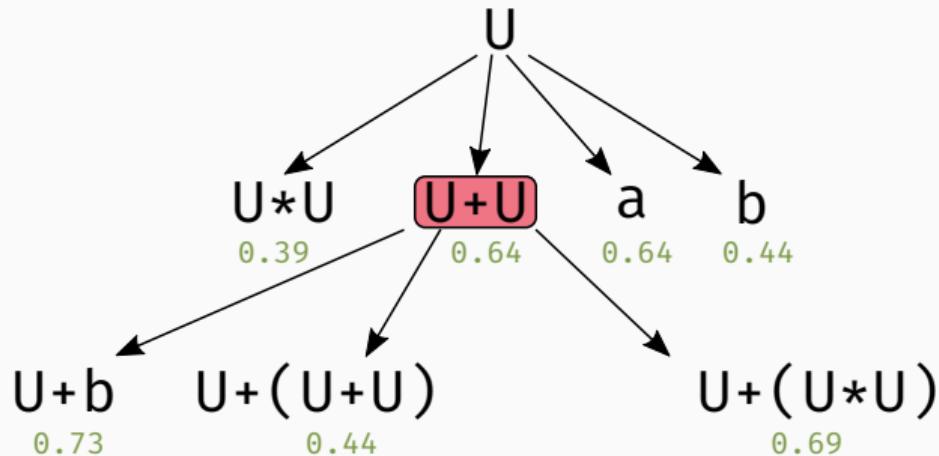


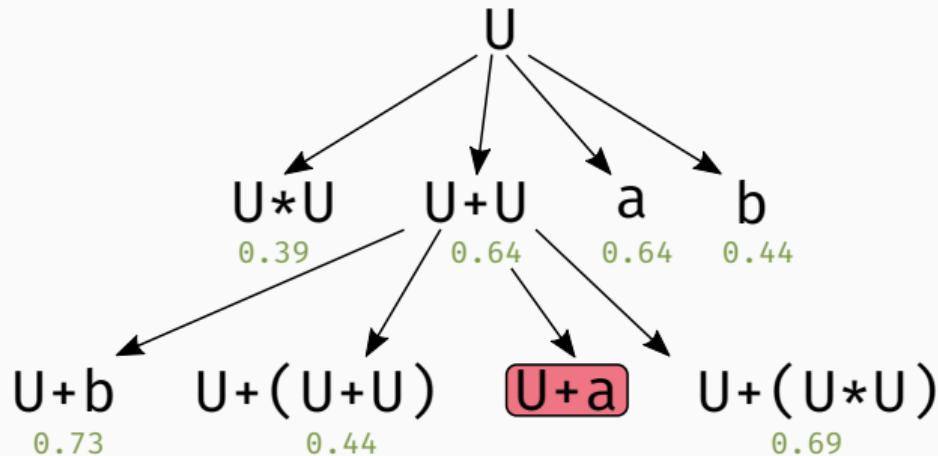


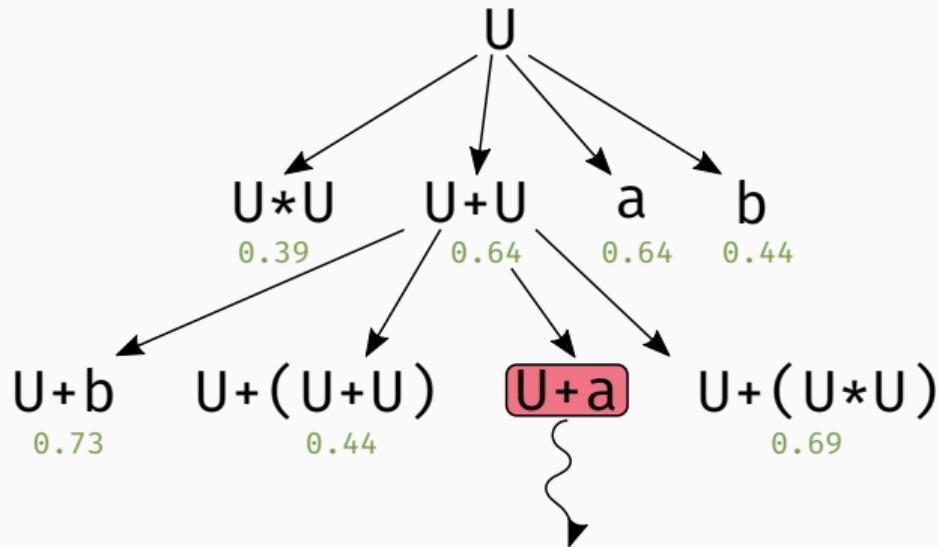


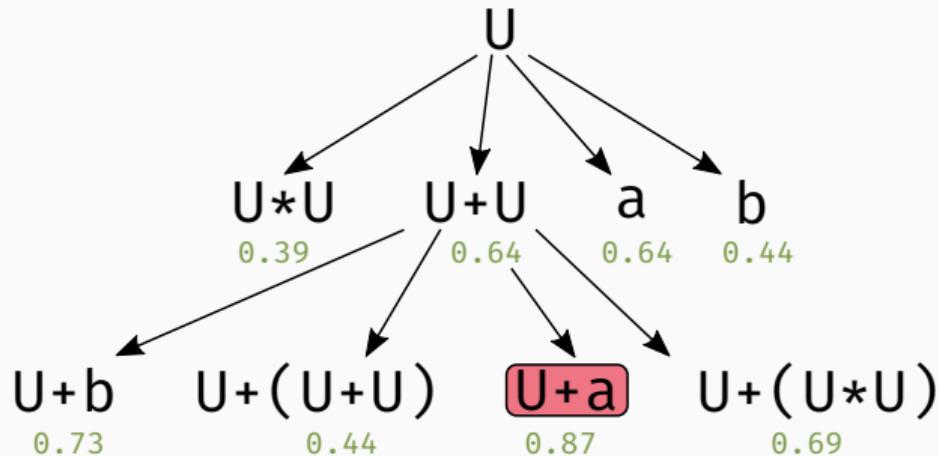


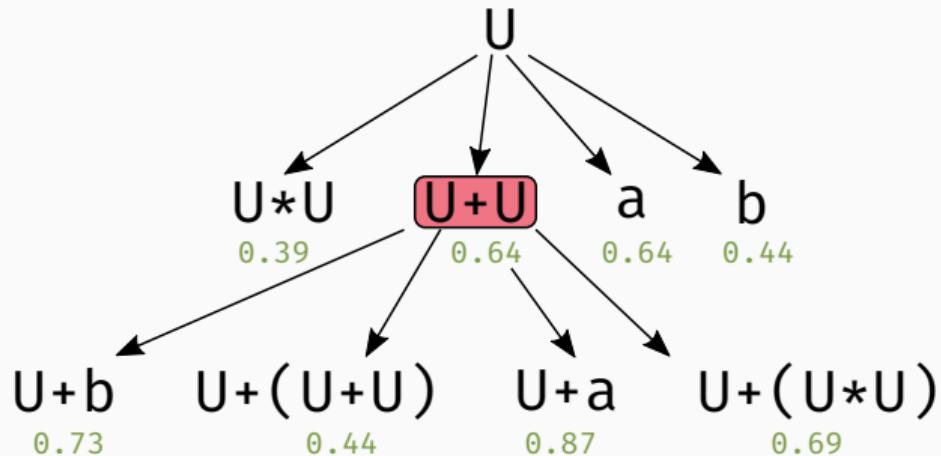


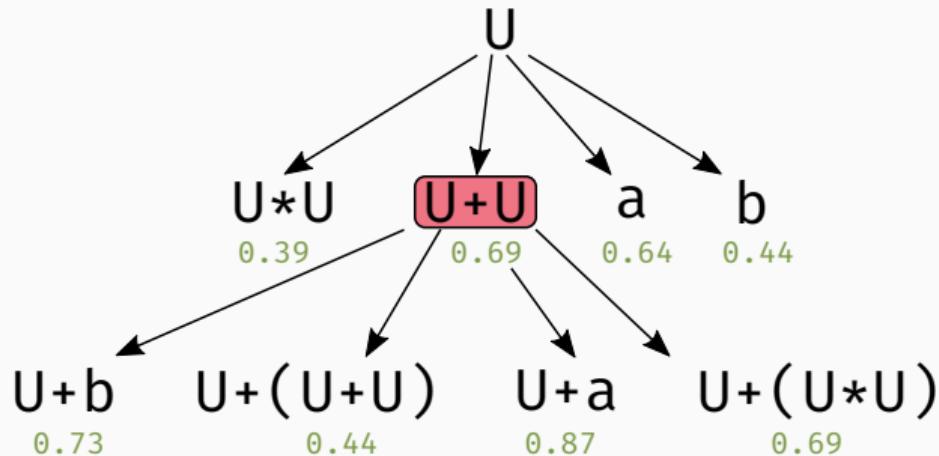


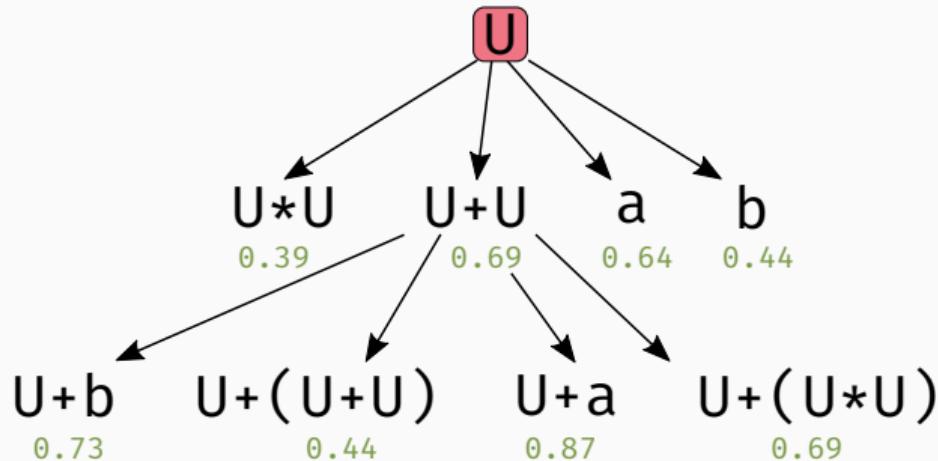


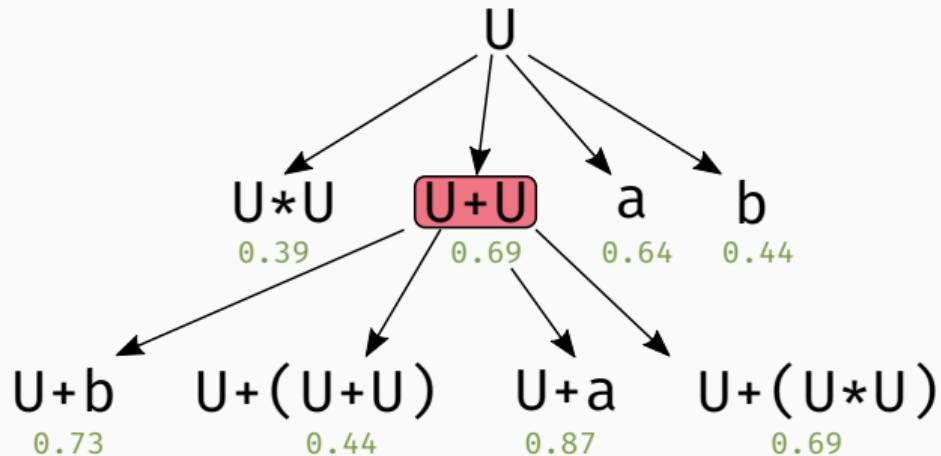


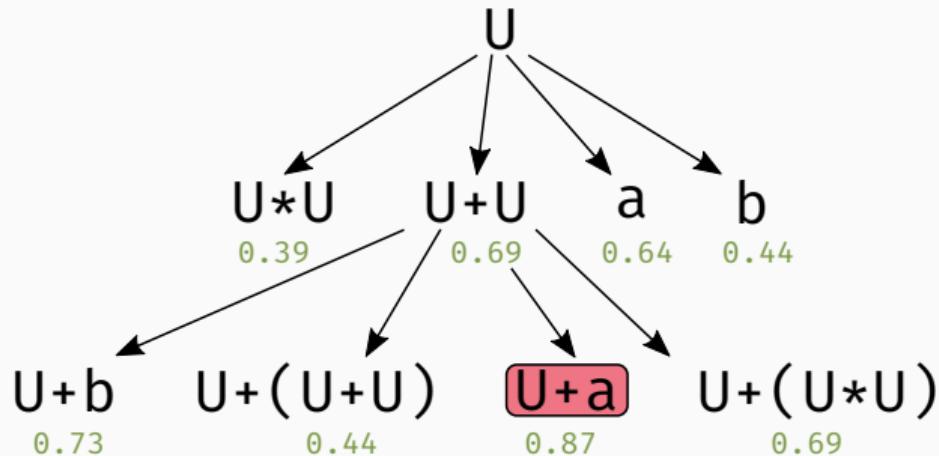


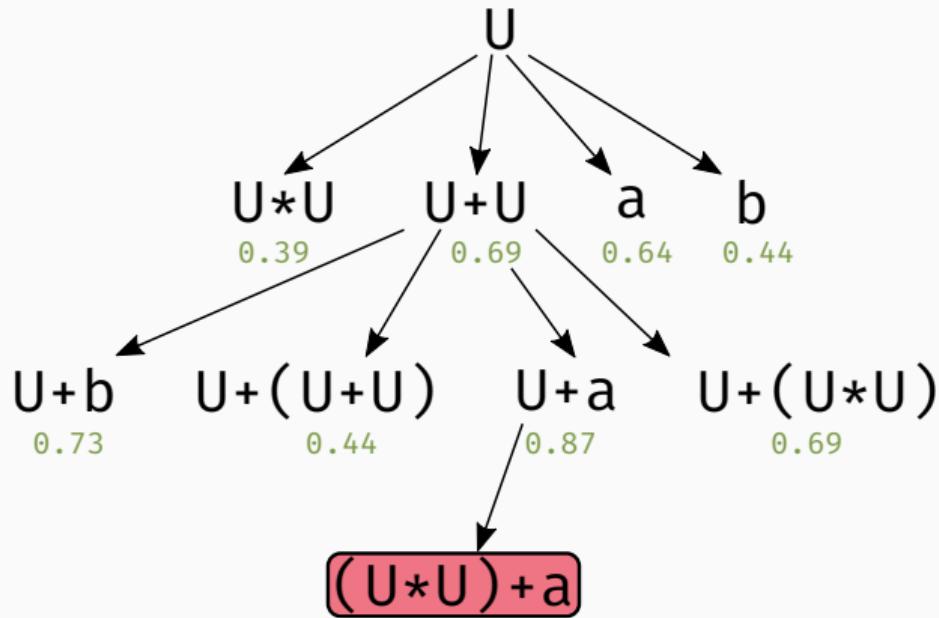


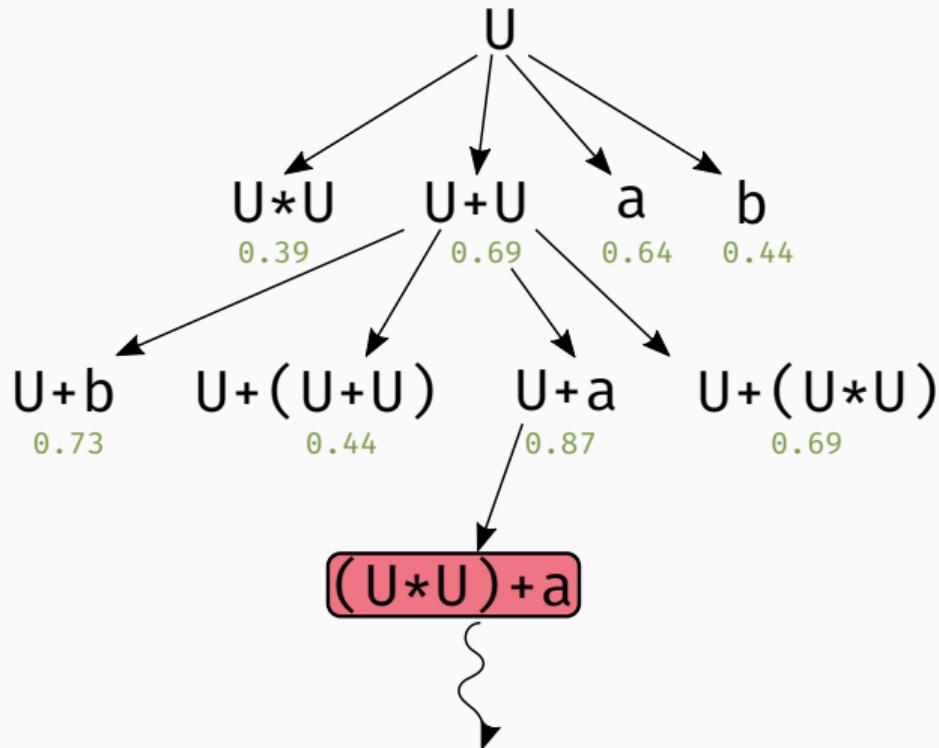


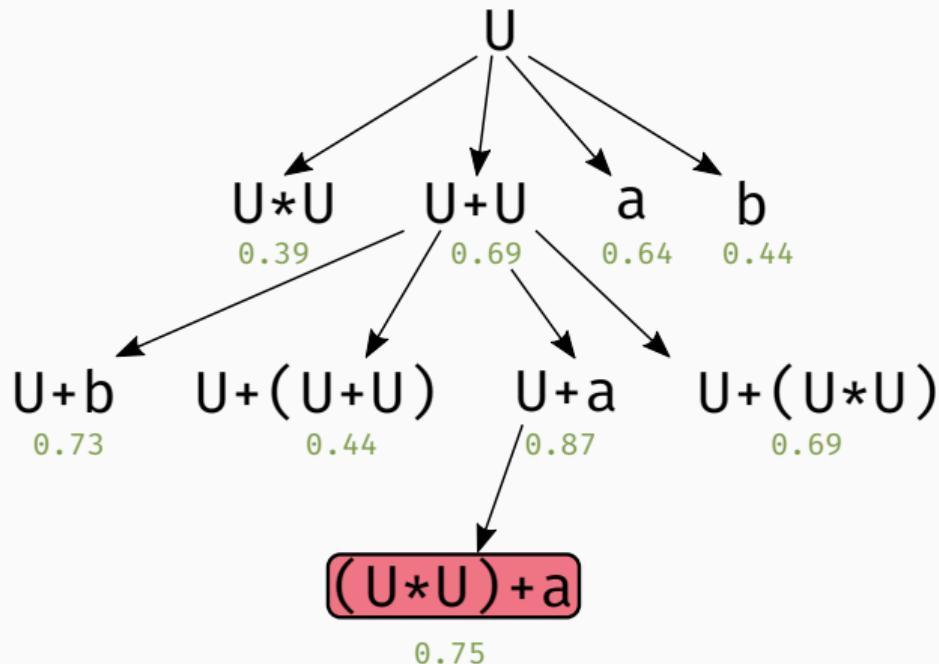


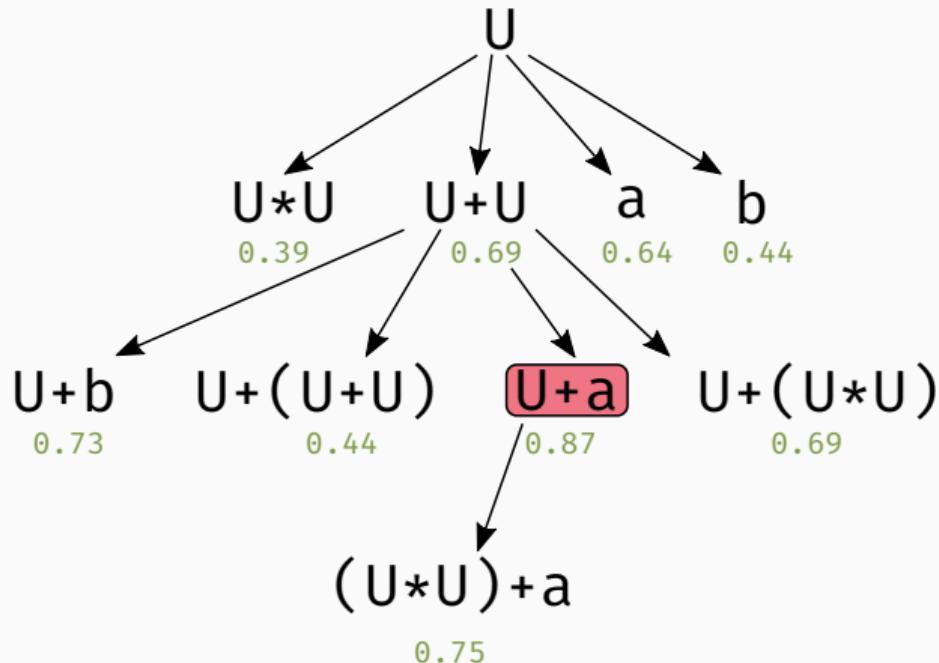


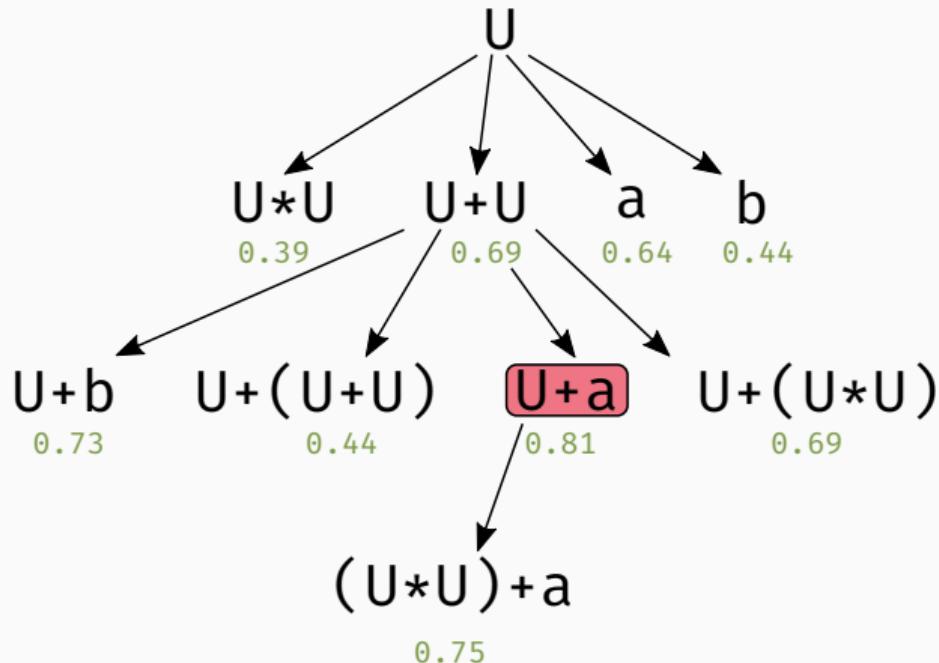


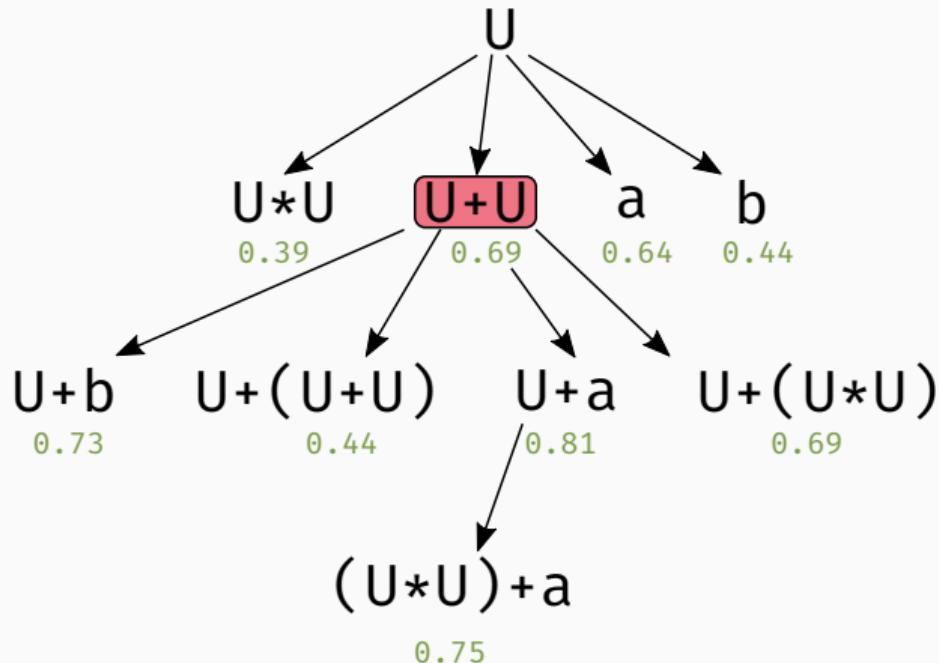


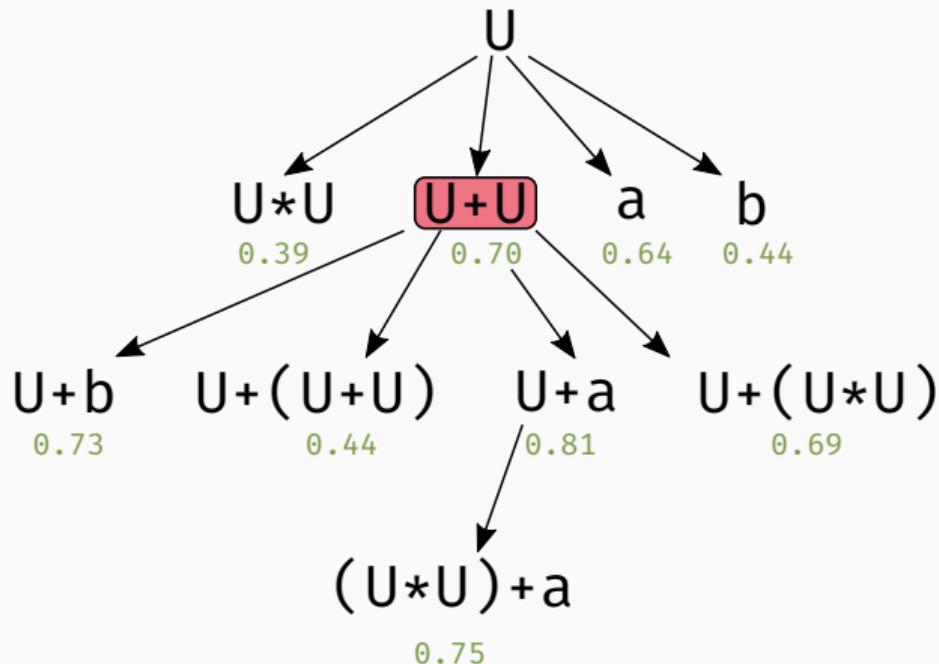


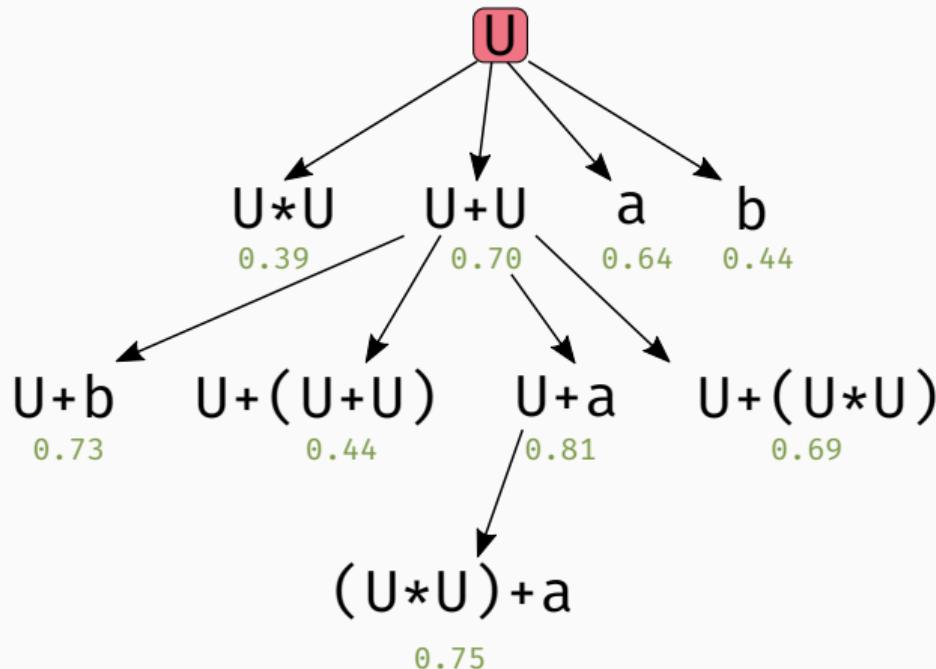


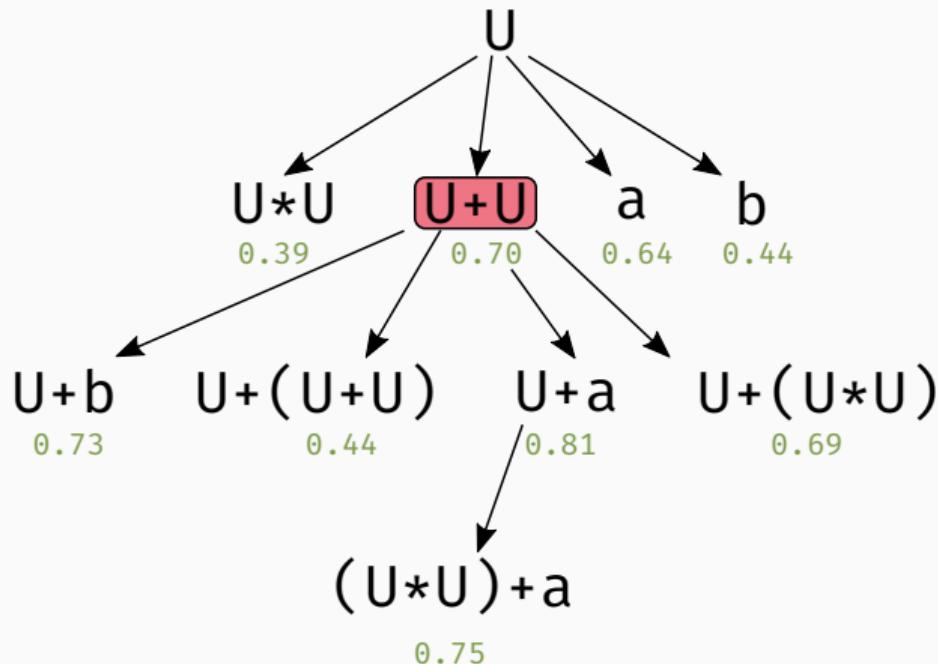


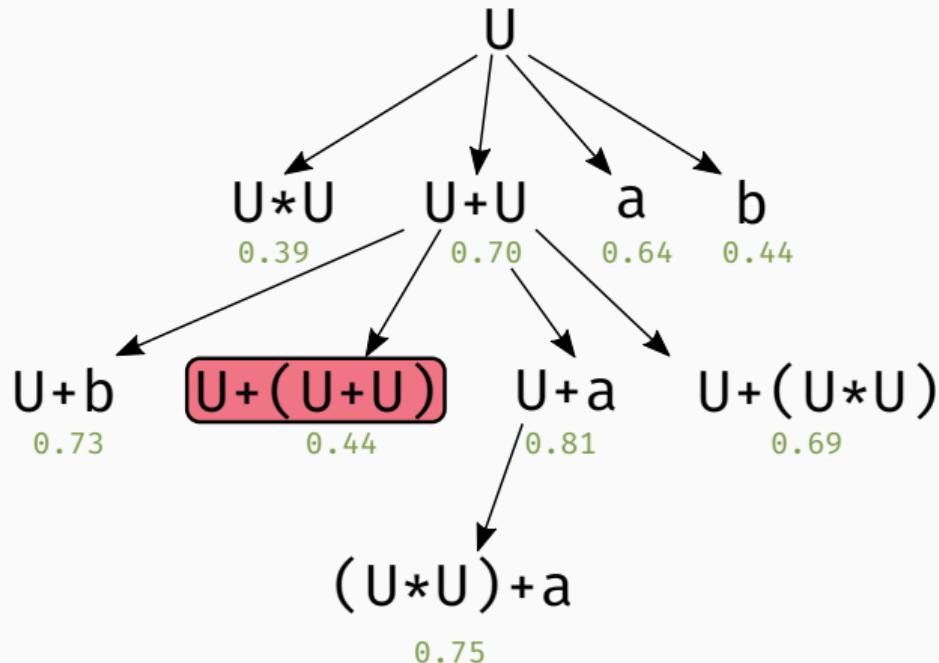


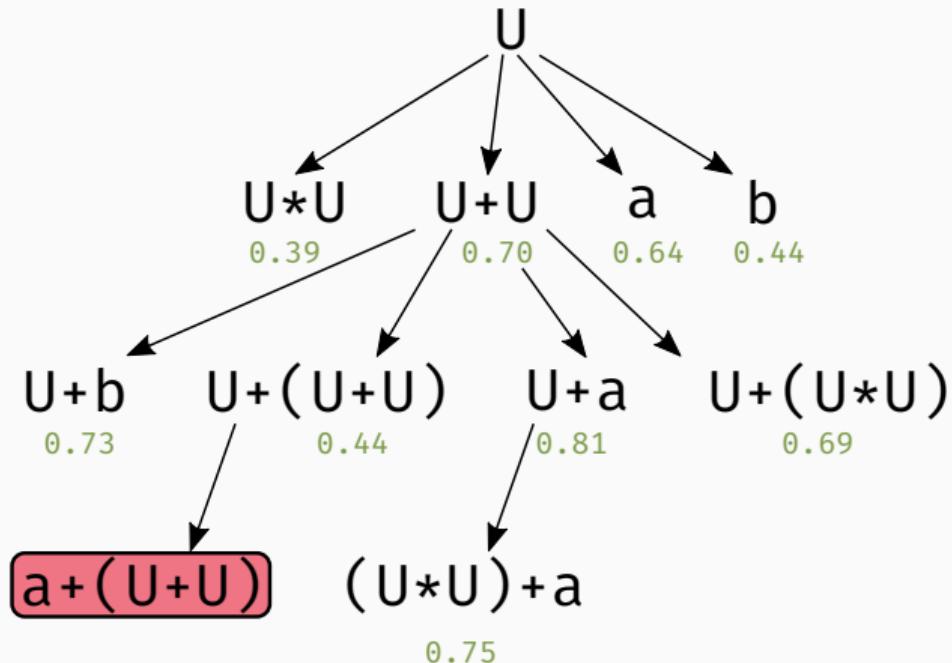


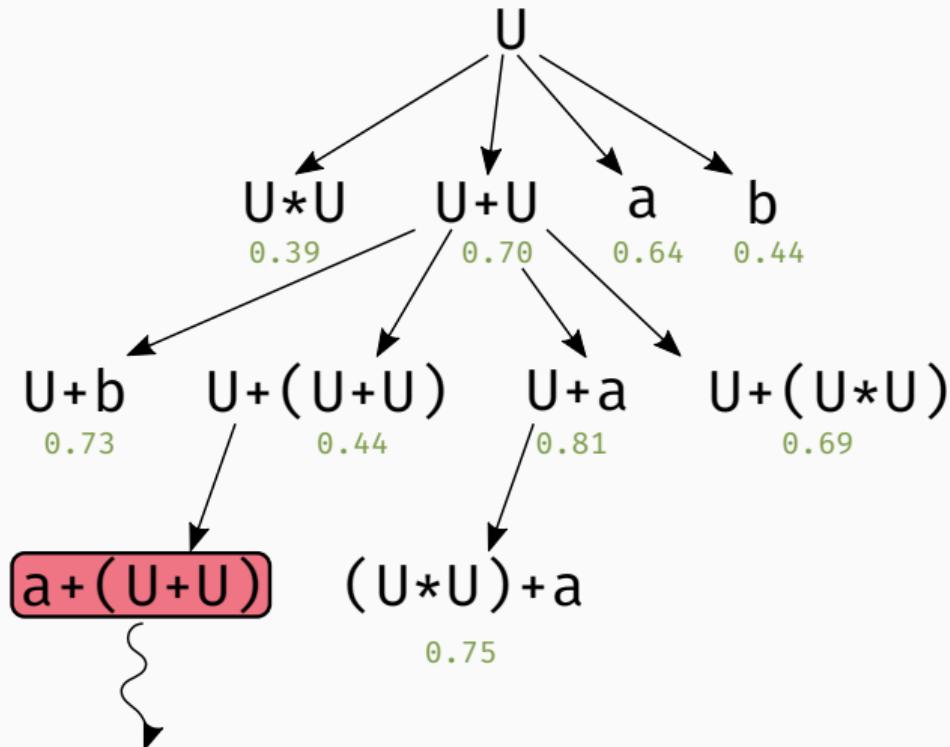


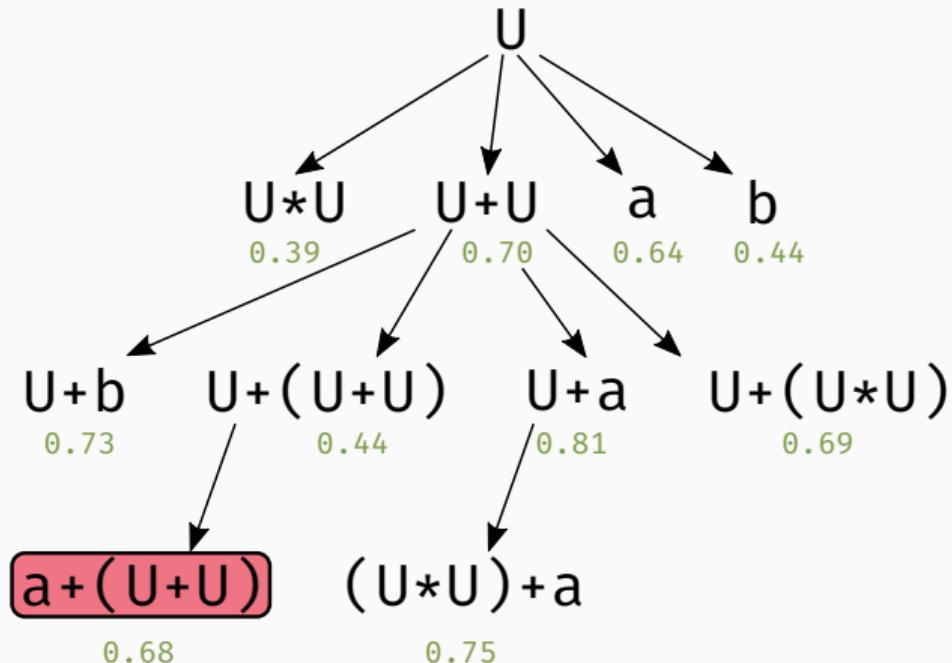


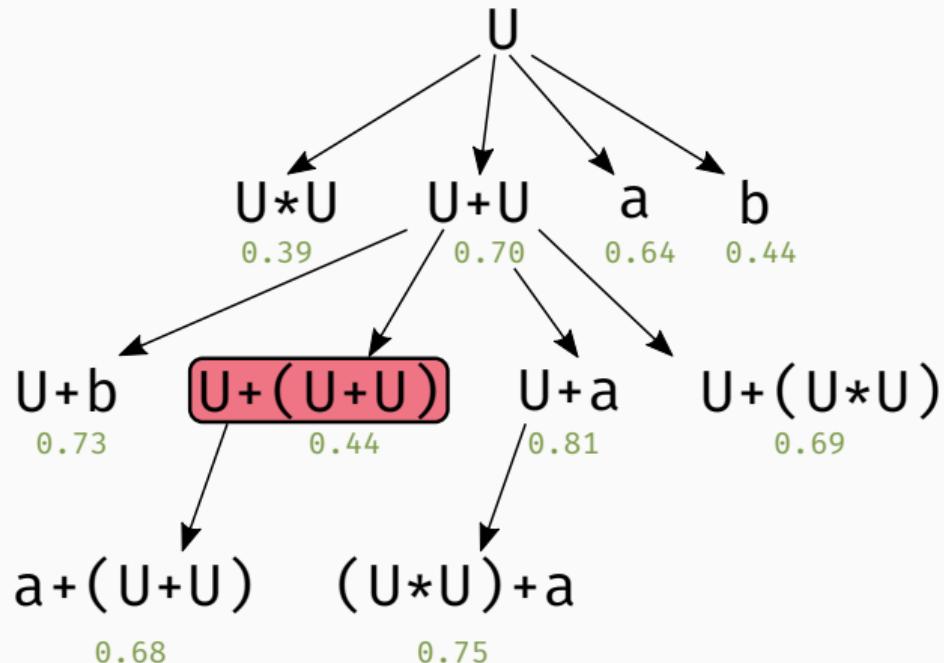


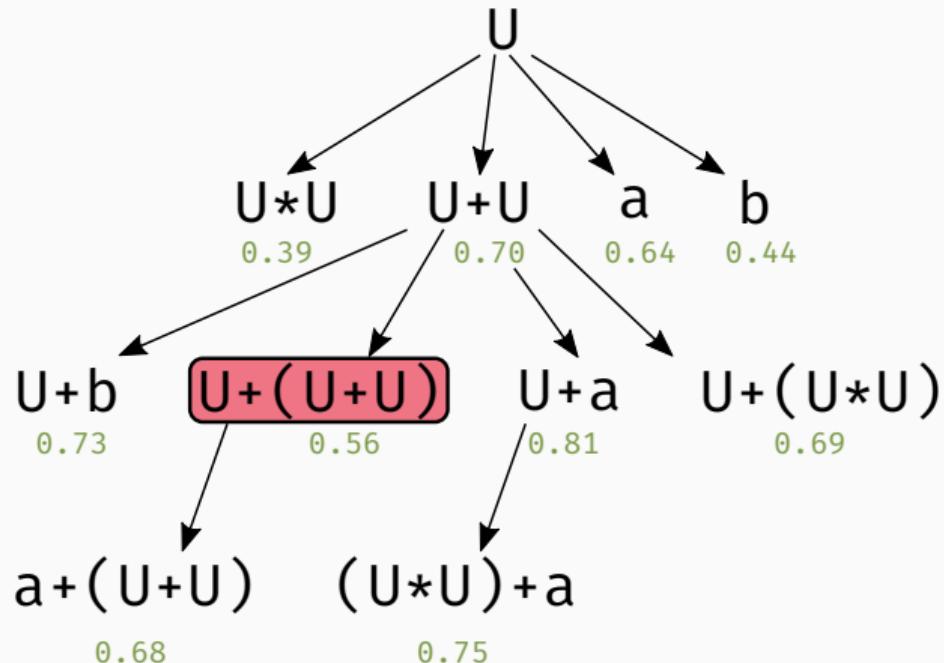


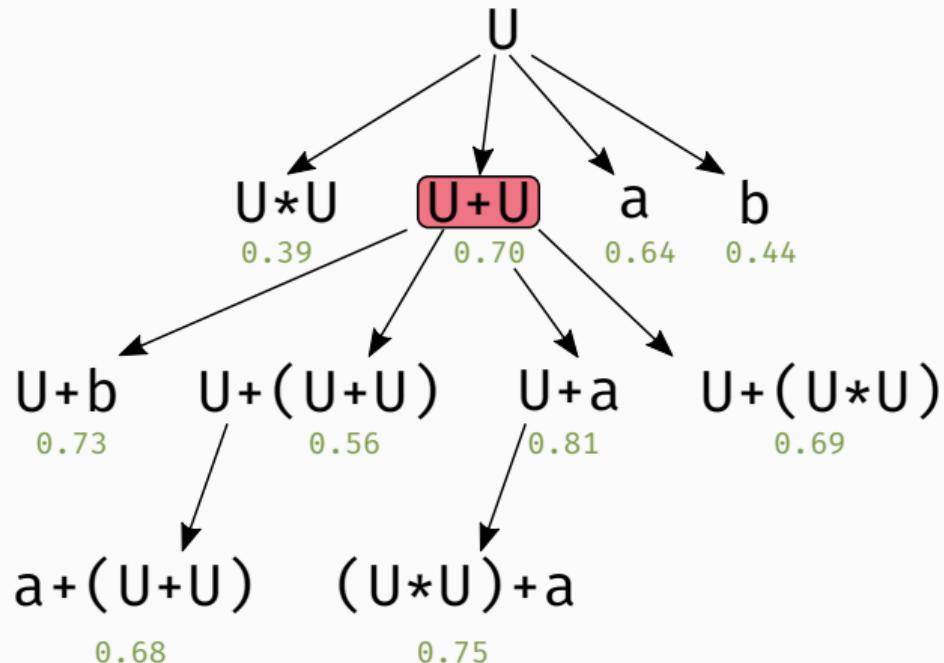


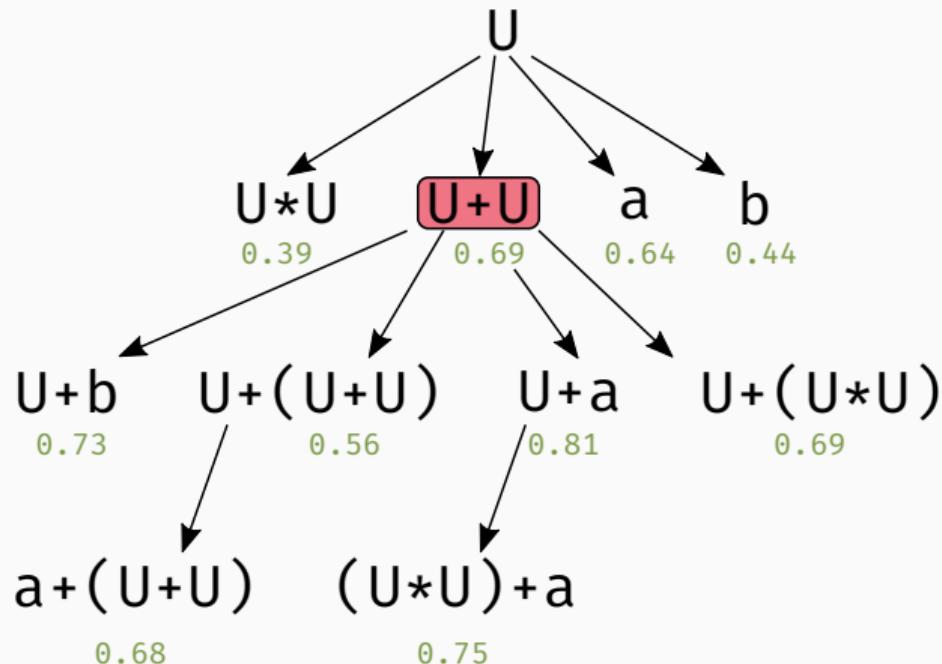


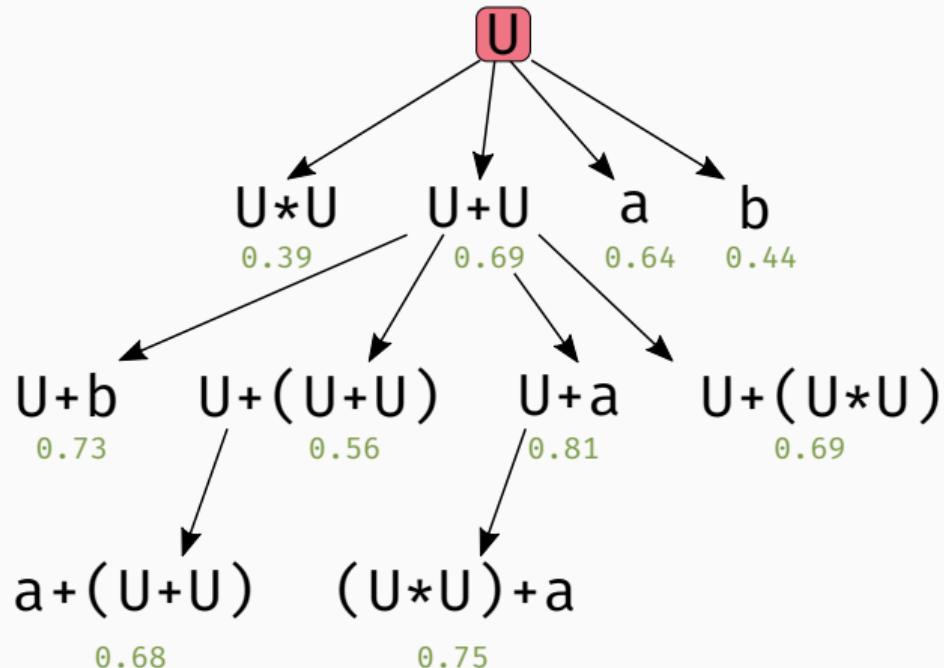


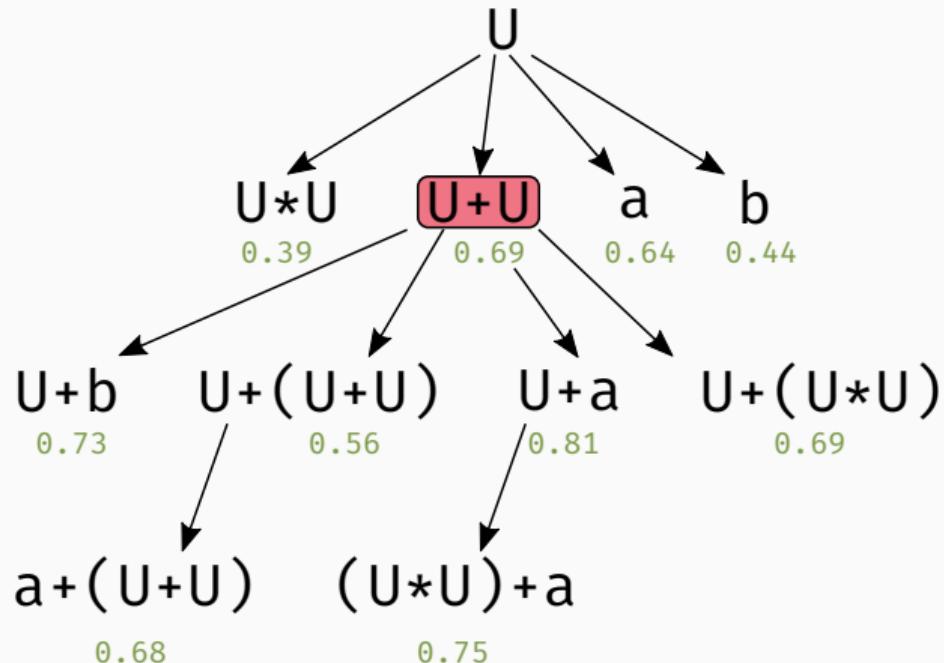


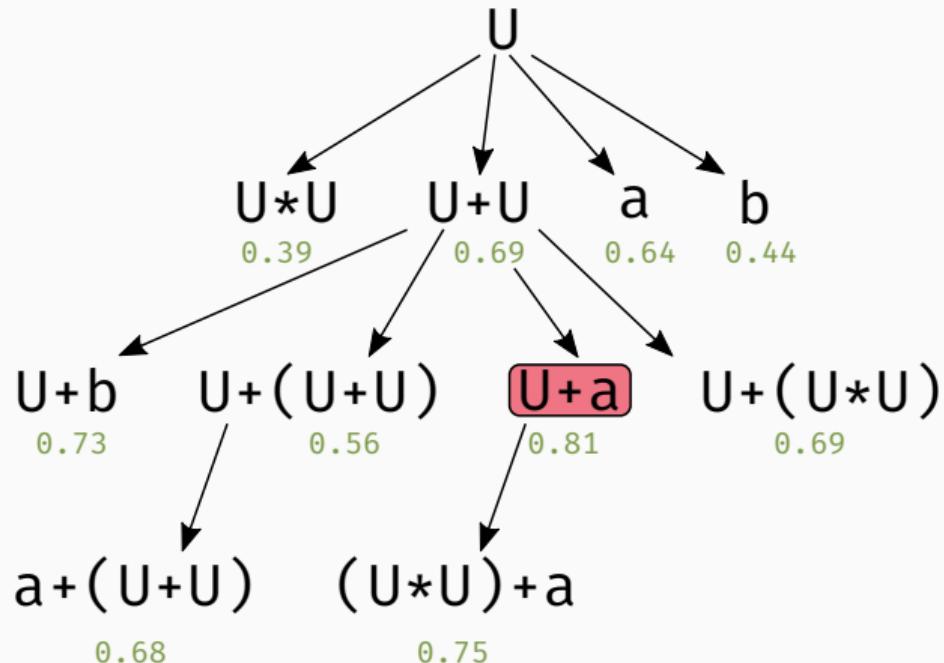


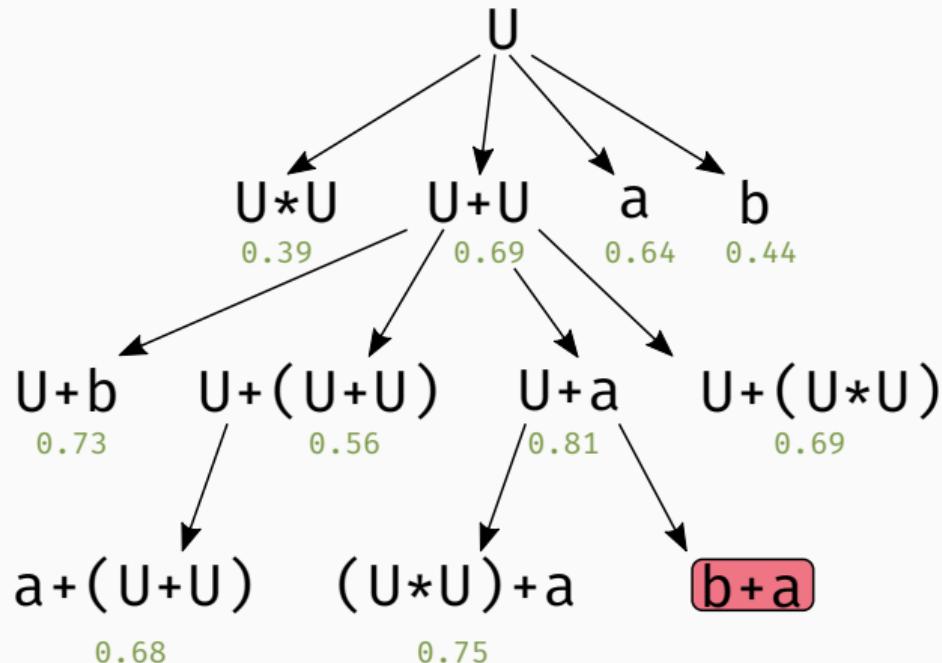


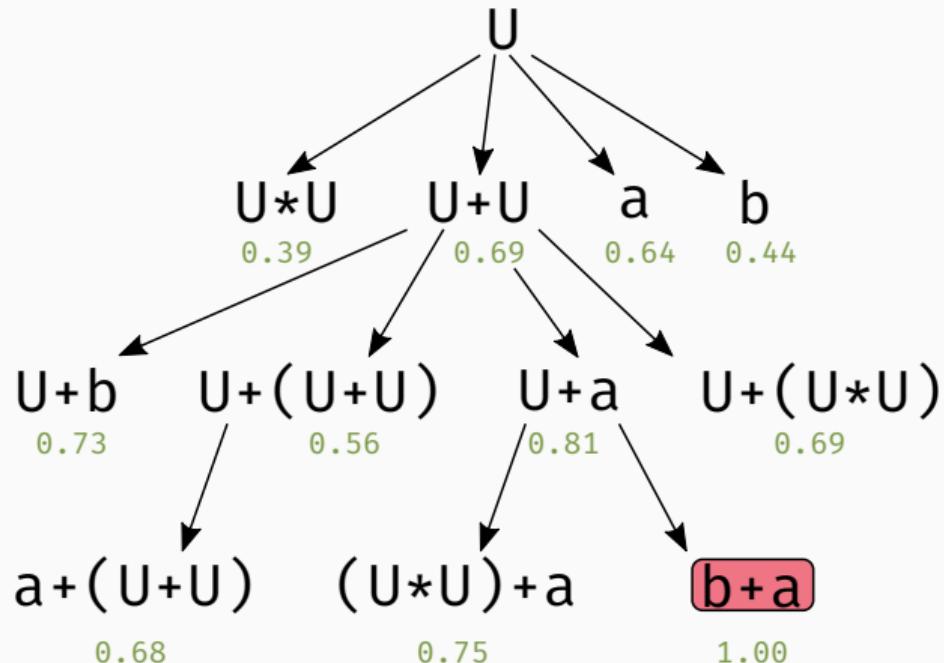


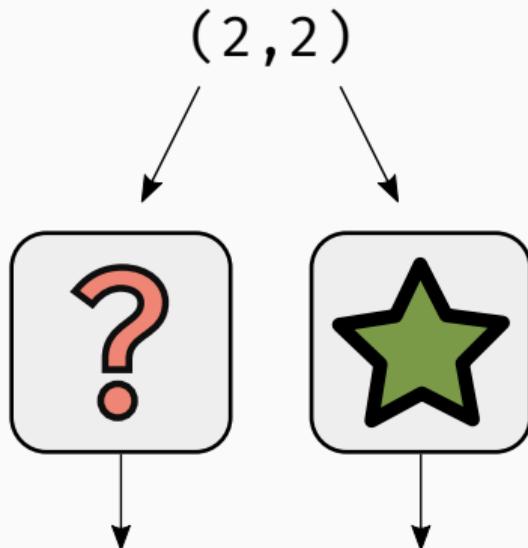


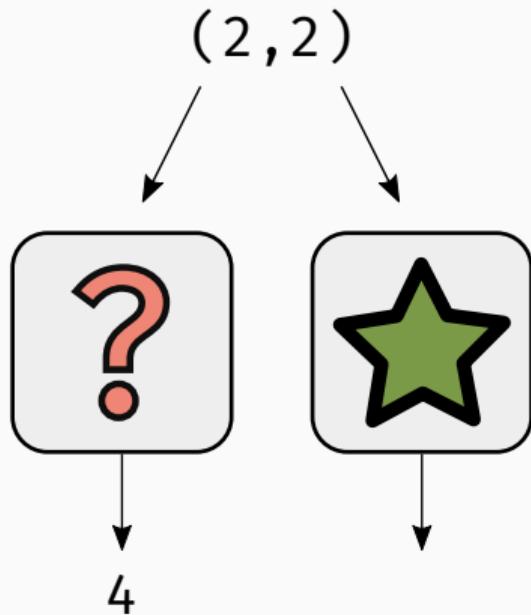


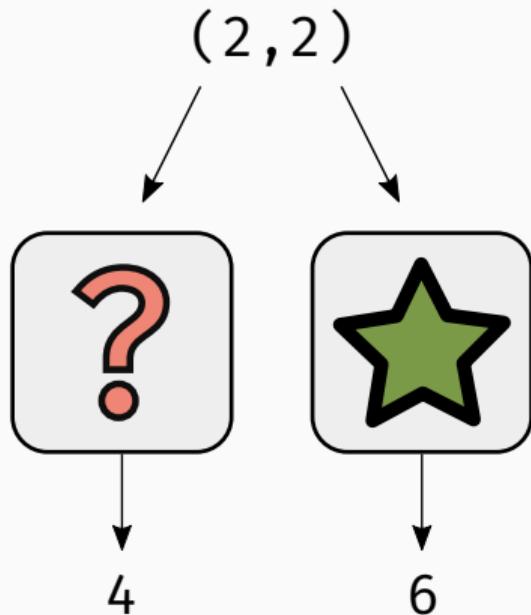


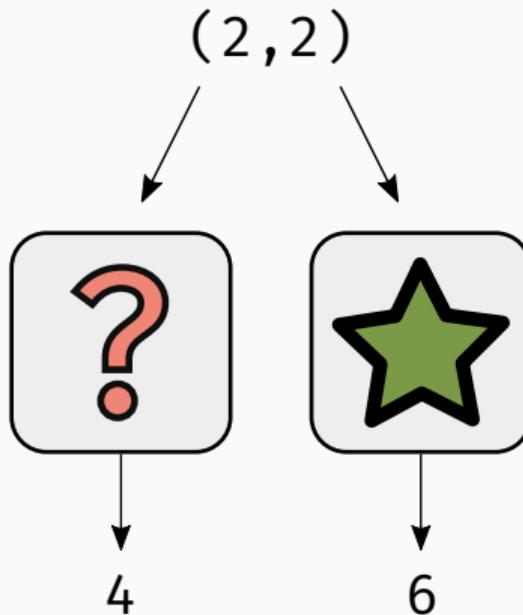




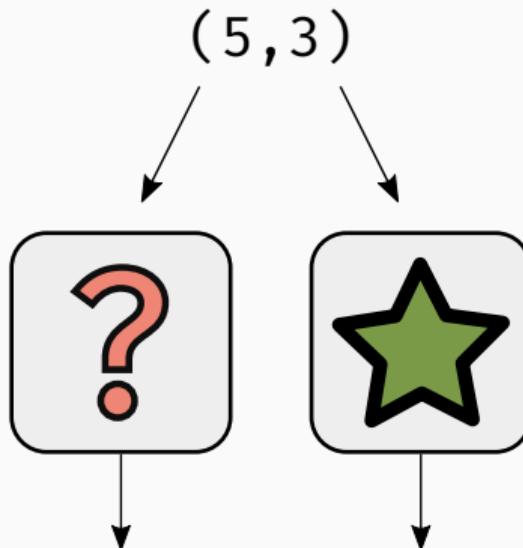




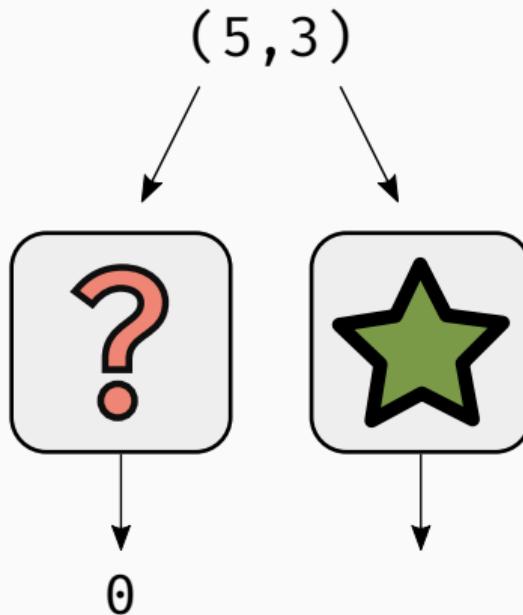




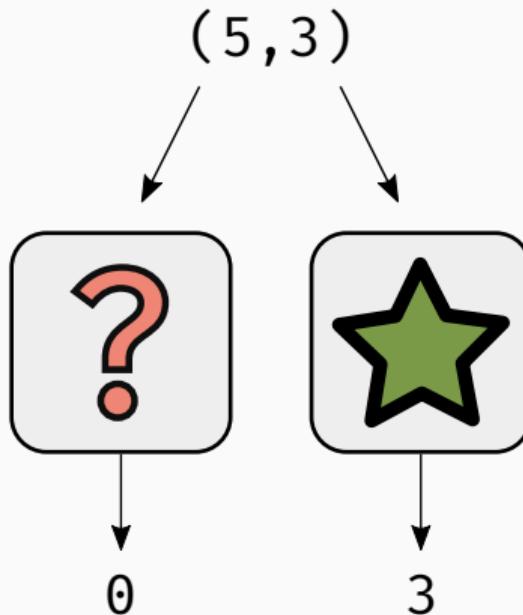
similarity(4, 6) = 0.78



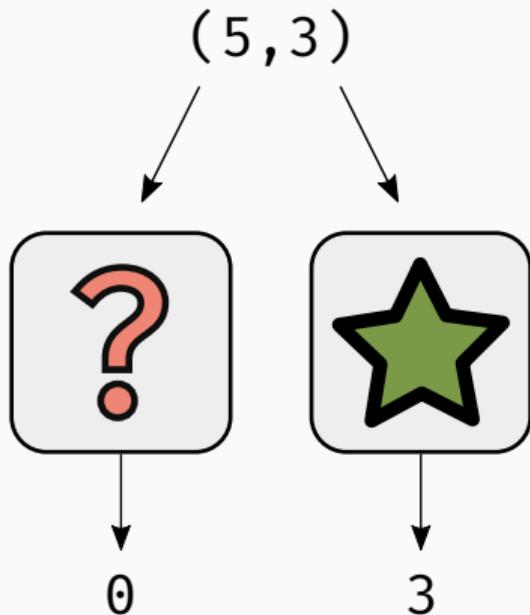
$\text{similarity}(4, 6) = 0.78$



$\text{similarity}(4, 6) = 0.78$

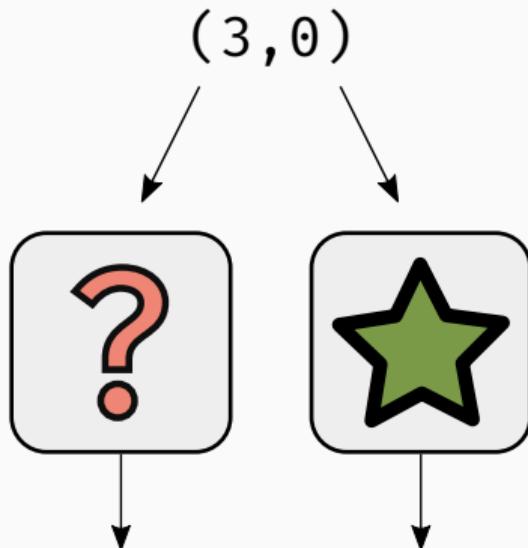


$\text{similarity}(4, 6) = 0.78$



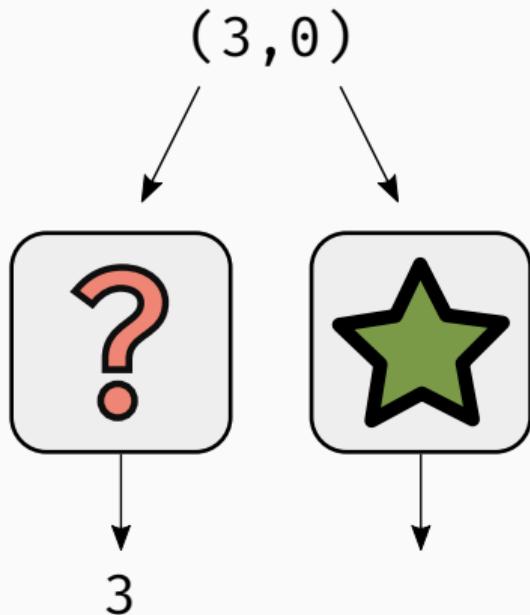
$$\text{similarity}(4, 6) = 0.78$$

$$\text{similarity}(0, 3) = 0.33$$



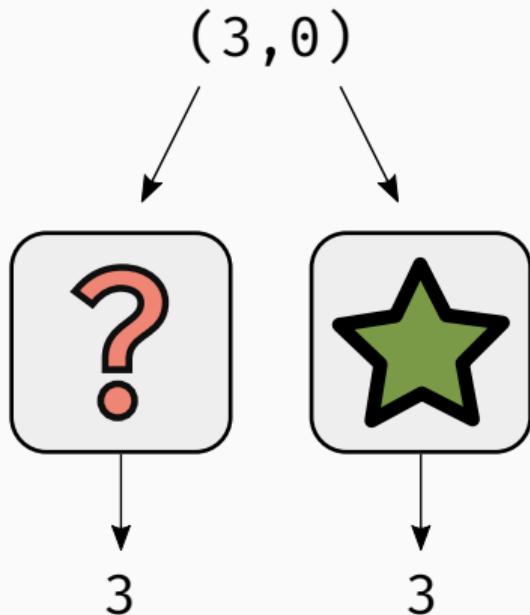
$\text{similarity}(4, 6) = 0.78$

$\text{similarity}(0, 3) = 0.33$



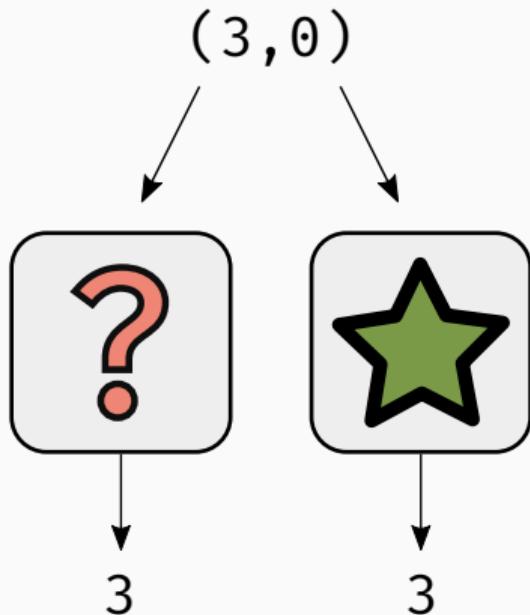
$\text{similarity}(4, 6) = 0.78$

$\text{similarity}(0, 3) = 0.33$



$\text{similarity}(4, 6) = 0.78$

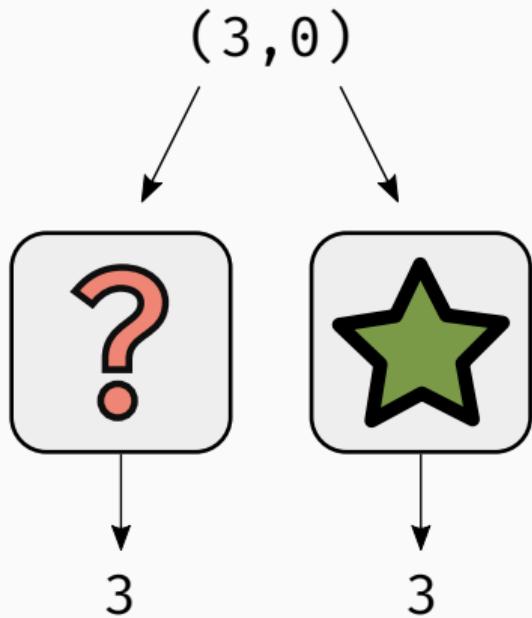
$\text{similarity}(0, 3) = 0.33$



$\text{similarity}(4, 6) = 0.78$

$\text{similarity}(0, 3) = 0.33$

$\text{similarity}(3, 3) = 1.0$



similarity(4, 6) = 0.78

similarity(0, 3) = 0.33

similarity(3, 3) = 1.0

average score: 0.70

11110111100100001000110010000000

11100010000110011110101100000000

Let's compare:

111	1011110010000100011001	00000000
111	00010000110011110110110	00000000

Are they in the same range?

11110111100100001000110010000000  
1110001000011001111010110000000

How many bits are different?

11110111100100001000110010000000  
00010101011101101010000110000000  
11100010000110011110101100000000

How close are they numerically?

# DEMO

How to synthesize obfuscated code?



static disassembly



static disassembly

```
54 68 69 73 20 64 6f
65 73 6e 27 74 20 6c
6f 6f 6b 20 6c 69 6b
65 20 61 6e 79 74 68
69 6e 67 20 74 6f 20
6d 65 2e de ad be ef
```

memory dump



static disassembly

```
54 68 69 73 20 64 6f
65 73 6e 27 74 20 6c
6f 6f 6b 20 6c 69 6b
65 20 61 6e 79 74 68
69 6e 67 20 74 6f 20
6d 65 2e de ad be ef
```

memory dump

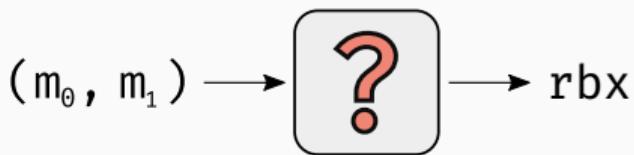
```
mov r15, 0x200
xor r15, 0x800
nov rbx, rbp
add rbx, 0xc0
nov rbx, qword ptr [rbx]
nov r13, 1
nov rcx, 0
nov r15, rbp
add r15, 0xc0
or rcx, 0x88
add rbx, 0xb
nov r15, qword ptr [r15]
or r12, 0xffffffff80000000
sub rcx, 0x78
novz r10, word ptr [rbx]
xor r12, r13
add r12, 0xffff
add r15, 0
nov r8, rbp
sub rcx, 0x10
or r12, r12
or rcx, 0x800
novz r11, word ptr [r15]
xor rcx, 0x800
nov r12, r15
add r8, 0
xor r12, 0xf0
nov rbx, 0x58
add r11, rbp
mov r15, rdx
xor r10d, dword ptr [r12]
sub r15, 0x800
or rdx, 0x400
mov rsi, 0x200
mov r14, rbp
sub rsi, rsi
mov rdi, rbp
sub rsi, r9
sub r8, rsi
add r14, 0
add rsi, rax
and r8, 0x88
xor rsi, r14
mov rsi, rbp
add rdi, 0xc0
sub r8, rdi
add r8, 0x78
add rsi, 4
mov rcx, 0x200
mov rdi, qword ptr [rdi]
add dword ptr [rsi], 0x254
xor rcx, 0xf0
add rcx, r10
add rdi, 6
mov r8, 0x400
mov ax, word ptr [rdi]
mov r8, 1
```

instruction trace

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

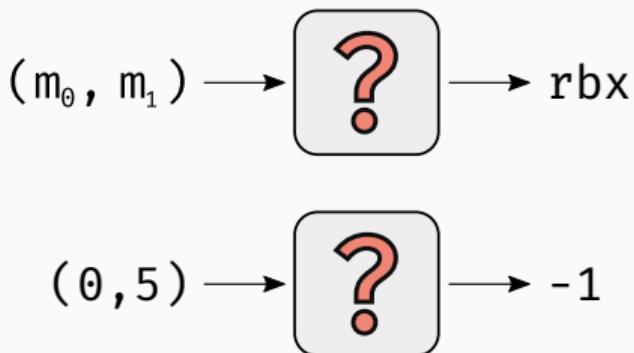
Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
• not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



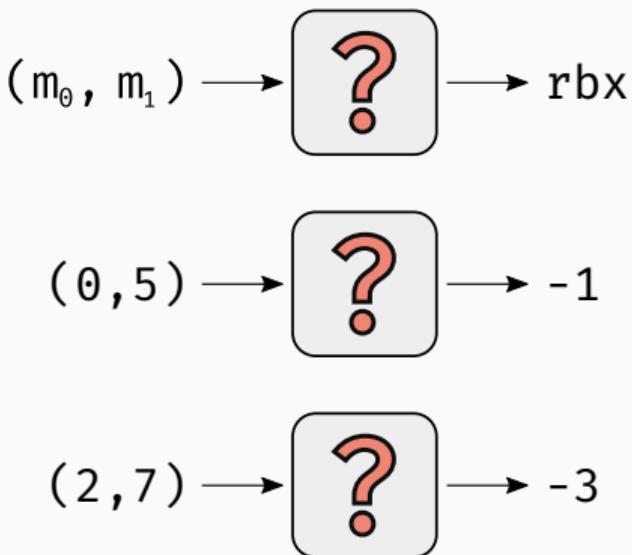
Handler performing `nor`  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
•   not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



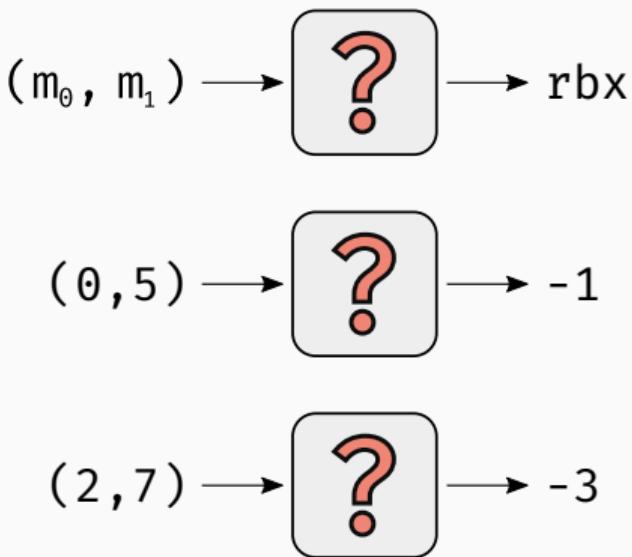
Handler performing nor  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
• not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



Handler performing nor  
(with flag side-effects)

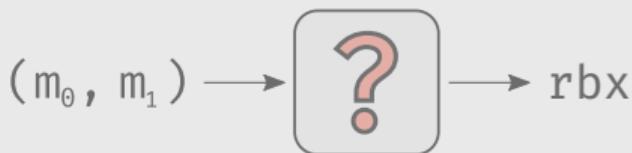
```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
• not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



Handler performing nor  
(with flag side-effects)

• • •

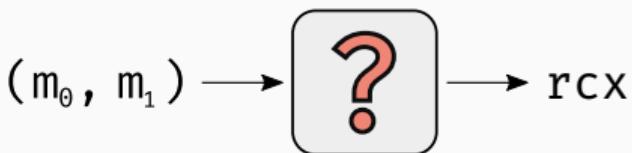
```
__handle_vnor:  
    mov rcx, [rbp]  
    mov rbx, [rbp + 4]  
    not rcx  
• not rbx  
    and rcx, rbx  
    mov [rbp + 4], rcx  
    pushf  
    pop [rbp]  
    jmp __vm_dispatcher
```



Handler performing nor  
(with flag side-effects)

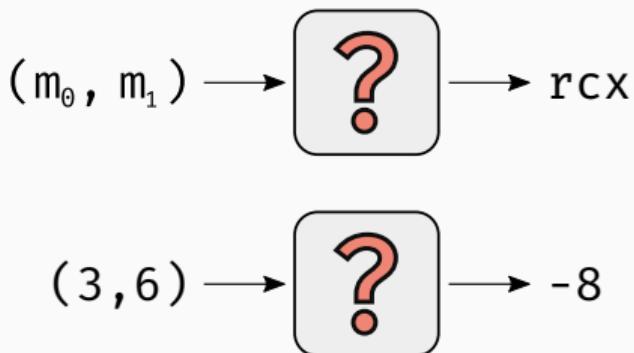
• • •

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
• and   rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



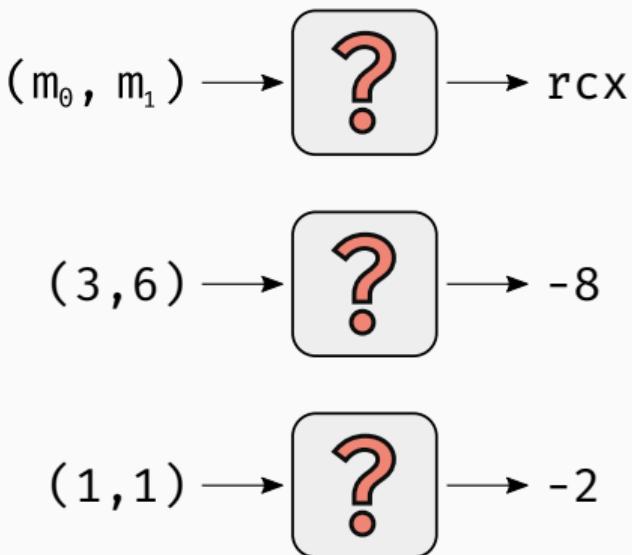
Handler performing **nor**  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
• and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



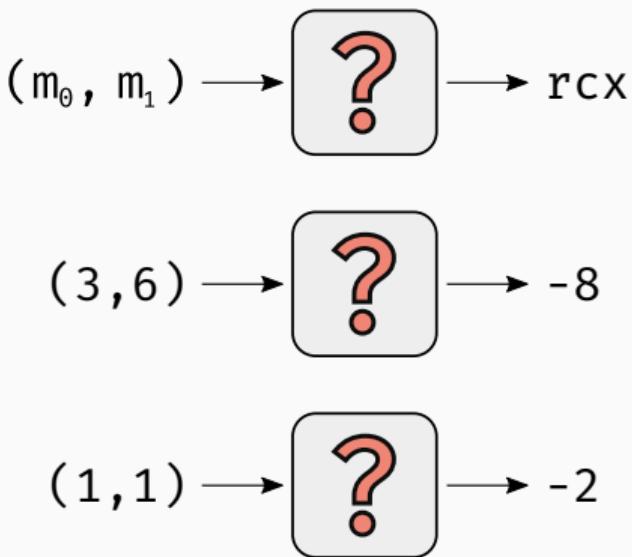
Handler performing nor  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
• and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



Handler performing nor  
(with flag side-effects)

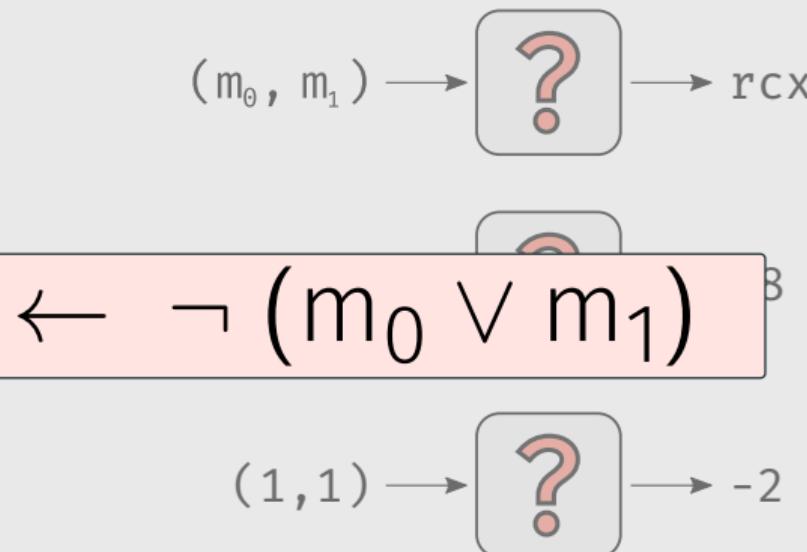
```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
• and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



Handler performing nor  
(with flag side-effects)

• • •

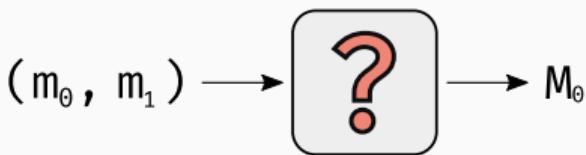
```
__handle_vnor:  
    mov rcx, [rbp]  
    mov rbx, [rbp + 4]  
    not rcx  
    not rbx  
• and rcx, rbx  
    mov [rbp + 8]  
    pushf  
    pop [rbp]  
    jmp __vm_dispatcher
```



Handler performing nor  
(with flag side-effects)

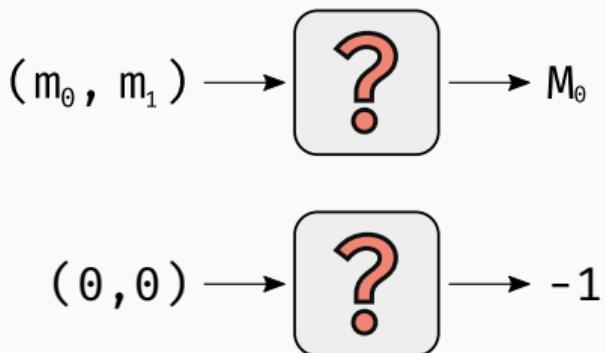
• • •

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
•   mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



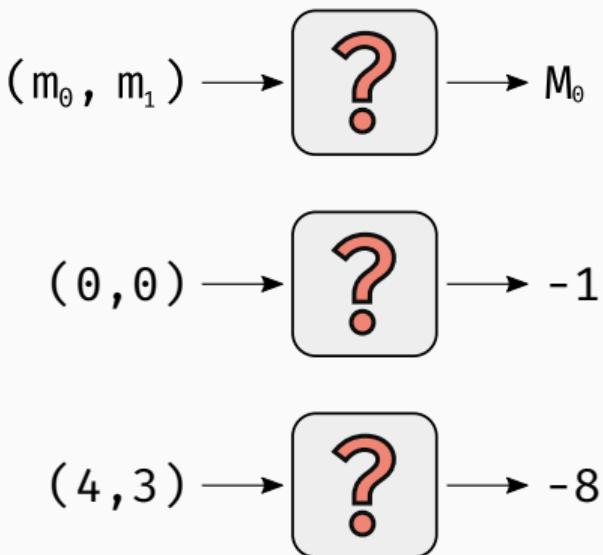
Handler performing nor  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
•   mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



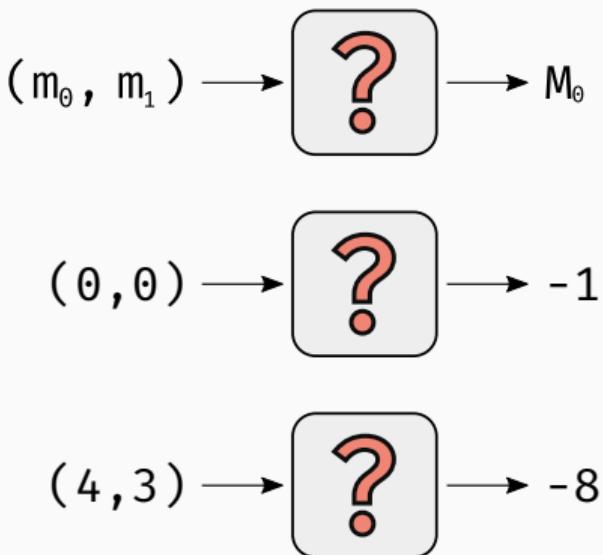
Handler performing nor  
(with flag side-effects)

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
•   mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



Handler performing nor  
(with flag side-effects)

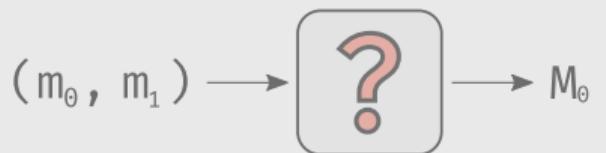
```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
•   mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



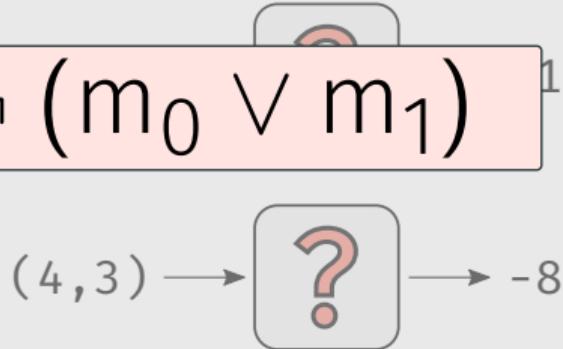
Handler performing nor  
(with flag side-effects)

• • •

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
•   mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```



$$M_0 \leftarrow \neg(m_0 \vee m_1)$$



Handler performing nor  
(with flag side-effects)

• • •

```
__handle_vnor:  
    mov rcx, [rbp]  
    mov rbx, [rbp + 4]  
    not rcx  
• not rbx  
• and rcx, rbx  
• mov [rbp + 4], rcx  
    pushf  
    pop [rbp]  
    jmp __vm_dispatcher
```

$$\text{rbx} \leftarrow \neg m_0$$

$$\text{rcx} \leftarrow \neg (m_0 \vee m_1)$$

$$M_0 \leftarrow \neg (m_0 \vee m_1)$$

Handler performing **nor**  
(with flag side-effects)

WinDbg



Valgrind

x64dbg



Unicorn

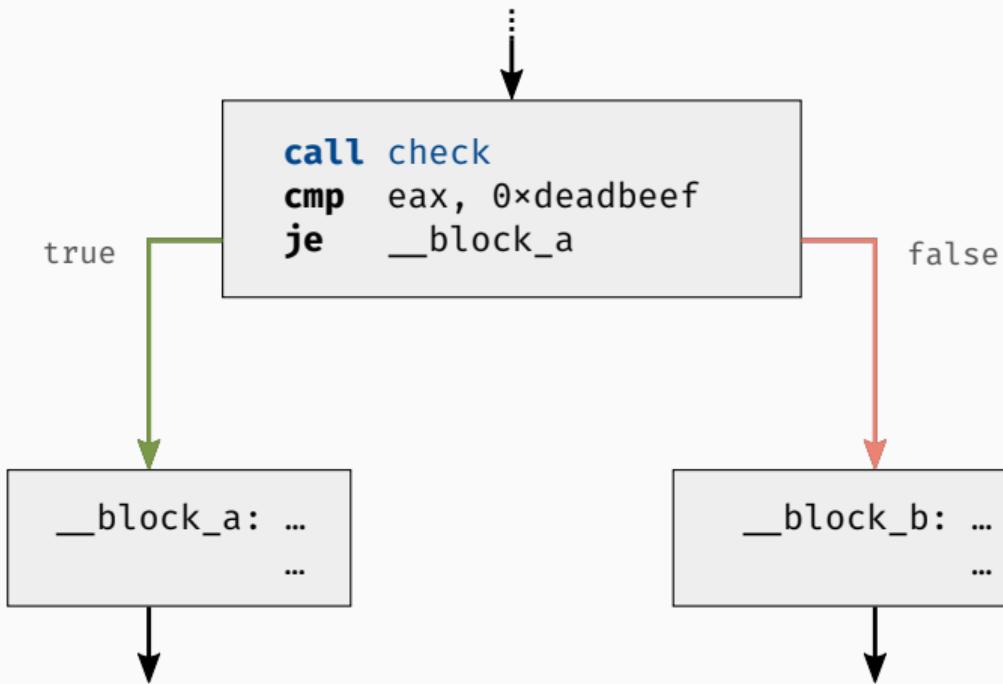
DynamoRIO

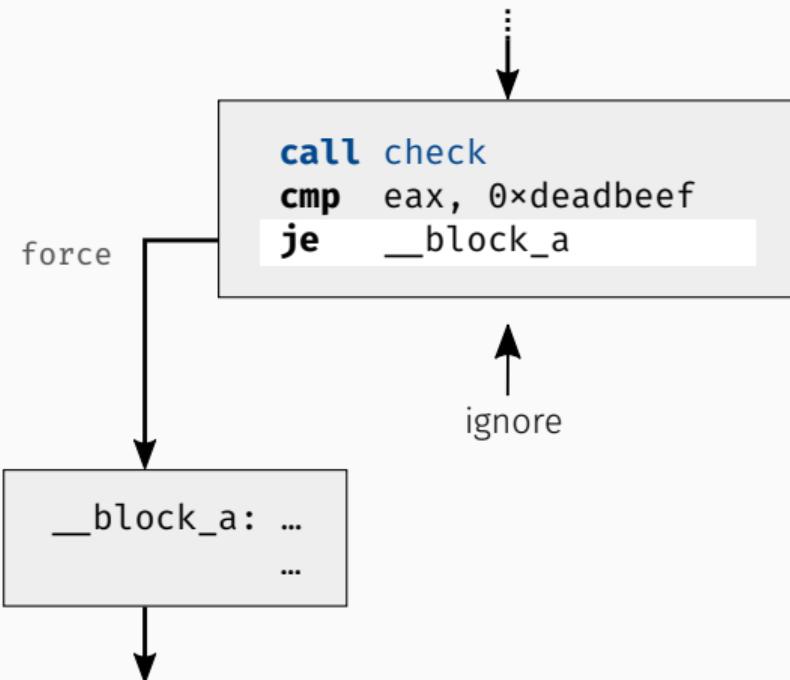


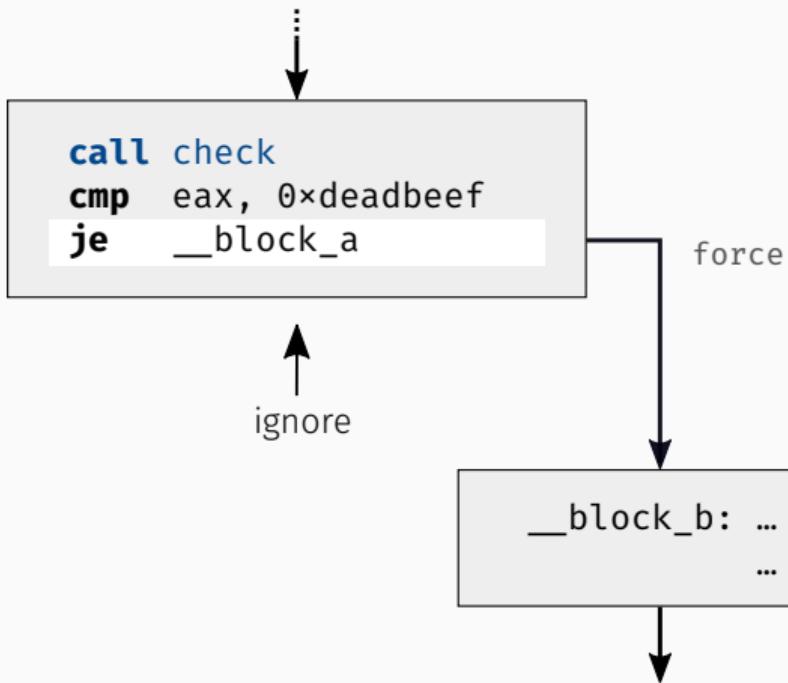
Miasm

<your tool here>

Metasm







- program synthesis framework for code deobfuscation
- written in Python
- random I/O sampling for assembly code
- MCTS-based program synthesis

<https://github.com/RUB-SysSec/syntia>

# DEMO

# Breaking Virtual Machine Obfuscation

**Hardening Technique #1** – Obfuscating individual VM components.

**Hardening Technique #2** – Duplicating VM handlers.

**Hardening Technique #3** – No central VM dispatcher.

**Hardening Technique #4** – No explicit handler table.

**Hardening Technique #5** – Blinding VM bytecode.

# #1: Obfuscating Individual VM Components

```

mov r15, 0x200 mov r15, rdx add r8, 1 or r14, r14 mov r14, 0x200 add r15, 0x3f
xor r15, 0x800 xor r10d, dword ptr [r12] add r8, 0x78 mov rax, rbp add rdx, 0xc0
mov rbx, rbp sub r15, 0x800 add word ptr [rbx], r10w and rcx, r13 add r11, r14
add rbx, 0xc0 or rdx, 0x400 mov r15, rax add rax, 4 and rsi, 9
mov rbx, qword ptr [rbx] nov rsi, 0x200 sub r15, rax add r8, -0x80000000
mov r13, 1 nov r14, rbp pop r9 mov r13, 0xfffff add rdx, 0xa
mov rcx, 0 sub rsi, rsi mov r15, rbp and rcx, 0x20 add r11, 0x78
mov r15, rbp nov rdi, rbp add r13, r15 mov r10, rbp
add r15, 0xc0 nov r8, 0x400 mov r15, qword ptr [rcx] add r10, r15 mov r8b, byte ptr [rdx]
or rcx, 0x88 sub rsi, r9 add rcx, 8 add r14, r8 cmp r8b, 0
add rbx, 0xb sub r8, rsi mov r10, word ptr [rcx] add r10, 0x89 add rax, 2
mov r15, qword ptr [r15] add r14, 0 mov r9, rbp xor word ptr [r10], si sub rsi, 4
or r12, 0xffffffff80000000 add rsi, rax add r9, 0 xor rdx, r11 and rsi, 9
sub r15, 0x78 and r8, 0x88 xor r10d, dword ptr [r9] mov rsi, rbp xor r11, 0x10
movzx r10, word ptr [rbx] xor r14, r14 sub rdx, rbx add rdx, 0xc0
mov r12, r13 nov rsi, rbp sub r13, 0xf0 and rax, 0x40 add r10, 0x58
add r12, 0xfffff add rdi, 0xc0 mov rsi, 0 or rbx, 0xf0 xor r9, 0
add r15, 0 sub r8, rdi sub r13, 0x20 add rsi, 0x5a add r10, 0x20
mov r8, rbp add r8, 0x78 mov rbx, rbp mov r8, r12 add eax, dword ptr [r9]
sub r15, 0x10 add rsi, 4 or r13, 0x88 movzx rsi, word ptr [rsi] sub r11, r8 xor r10, 0x40
or r12, r12 nov rcx, 0x200 and rcx, 8 mov rax, 0x200 add rdx, 0x40
mov r15, 0x800 mov rdi, qword ptr [rdi] mov r8, 0x58 mov r14, rbp add r15, 0x88
movzx r11, word ptr [r15] add dword ptr [rsi], 0x2549b044 add rbx, 0xc0 and rax, rdx
mov r15, 0x800 xor r15, 0x800 mov r15, qword ptr [rbx] and rcx, 0x20 add r12, rbp
mov r12, r15 add r10, r10 sub r15, 0x20 add r14, 0x89 add rdi, 0x90
add r8, 0 add rdi, 6 add r13, 0x80 or rax, 0x40 add r12, 0
xor r12, 0xf0 mov r8, 0x400 sub r13, 0x10 xor si, 0x7a28 add dword ptr [r12], eax
mov rbx, 0x58 mov ax, word ptr [rdi] add rbx, 8 add rdx, 0x78
add r11, rbp mov r8, 1 mov si, word ptr [rbx] add rdx, 0x20 add r10, 0x20
xor r15, 0x800 mov rsi, rbp or r9, 0xfffff movzx r14, word ptr [r14]
and r12, 0x20 and r10, r10 sub r9, 1 mov rax, 0x58 add r13, r8
add rbx, 0x800 sub r15, 1 mov r9, rbp xor rdx, r12 add r10, 0x68
mov r11, qword ptr [r11] mov r15, rdi mov r12, 0x58 mov r9, 0x12
add rbd, 1 add rsi, 0x29 add r12, 0x58 xor rax, rdx mov r11, 0
and r12, r9 or r15, r8 sub r13, 0x80 add r8, 0x80 add r13, r8
mov rdx, 1 mov r8, rsi mov r15, r13 add r15, rsi and r15, 0x78
xor r10d, dword ptr [r8] add r14, r14 mov r14, rbp or rbx, 1
pushfd r9, r11 mov r13b, byte ptr [rsi] or r14, r12 add r14, r15 shl rax, 3
xor r15, 0xf0 cmp r13b, 0xd2 mov r10, rbp mov r8, 0
add r10, r10 and r10, 0xcc xor r15, r15 add r14, 0x29 add r8, rax
and rdx, r8 and r15, 0x20 xor r15, r15 xor r14, rdi sub r15, 0x10
mov r12, rbp or r15, r13 xor esi, dword ptr [r10] add qword ptr [rsi], r14 or byte ptr [r14], r10b
xor r15, 0x20 or r15, 4 add pushfq xor r11, r14 mov r10, 0x58
sub r15, 4 or r15, 4 add r13, 0x90 xor r11, r14 mov r8, rbp
add r11, 0x2549b044 sub r15, 0x400 mov r14, rsi sub r13, r8
or r15, 0x78 add r10, r8 add r14, rsi add r8, 0x127
and rdx, r10 add r15, 0x88 mov r8, 0 add rdx, 0xc0
mov rax, 0 add rdx, rbp xor r15, 0x3f add qword ptr [rdx], 0xd
add r12, 0x42 add r15, 0x5a mov r13, r15 xor r13, 0x40 jmp rbx

```

# #1: Obfuscating Individual VM Components

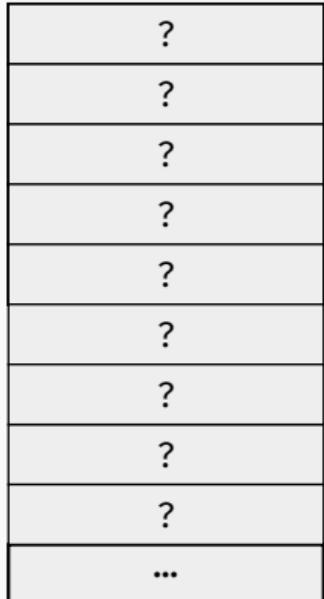
```

mov r15, 0x200      mov r15, rdx      add r8, 1        or r14, r14      mov r14, 0x200      add r15, 0x3f
xor r15, 0x800      xor r10d, dword ptr [r12]  or rax, rbp      add rdx, 0xc0
mov rbx, rbp       sub r15, 0x800      add word ptr [rbx], r10w  and rcx, r13      and r11, r14
add r15, qword ptr [rbx]  or rdx, 0x400      mov r15, rax      add rax, 4        or r15, 0x88
mov r15, 0xc0       or rsi, 0x200      sub r15, rax      add r13, 0xfffff
or r15, qword ptr [rbx]  mov rsi, r14      pop r9         add r13, 0x20
mov r13, 1          sub rsi, rsi      mov rcx, rbp      and r10, rbp
mov rcx, 0           mov rdi, rbp      add r10, rbp      mov r8b, byte ptr [rdx]
and r15, r14        sub rsi, r9       mov rcx, qword ptr [rcx]  add r11, 0x78
add r15, 0xc0       mov r8, 0x400      add r13, r15      cmp r8b, 0
or r15, 0x88        sub r8, r9       add rcx, 8        add r14, r8
add r15, 0xb          sub r8, rsi      movzx r10, word ptr [rcx]  je 0x49e
mov r15, 0xb          sub r8, rsi      add r10, 0x89      mov rdx, rbp
mov r15, qword ptr [r15]  add r14, 0        xor word ptr [r10], si  or r11, 0x40
or r12, 0xfffffffffb0000000  add rsi, rax      xor rdx, r11      and r15, 1
sub rcx, 0x78        and r8, 0x88      xor r10d, dword ptr [r9]  mov rsi, rbp
movzx r10, word ptr [rbx]  xor rsi, r14      and rdi, 0xfffffffffb0000000  xor r11, 0x10
xor r12, r13        mov rsi, rbp      sub rdx, rbx      add rdx, 0xc0
add r12, 0xfffff    add rdi, 0xc0      and r13, 0xf0      and rax, 0x40
add r15, 0          sub r8, rdi      mov rsi, 0        or rbx, 0xf0
mov r8, rbp       add r8, 0x78      sub r13, 0x20      mov r15, 0x12
sub rcx, 0x10        add rsi, 4        add r8, r5a      mov rdx, qword ptr [rdx]
or r12, r12        mov r8, 0x200     mov rbx, rbp      add r11, r8
or r15, 0x800        add rdi, qword r14, r14      mov r8, rdx
movzx r11, word ptr [r15]  add rdi, 0x40      mov r15, 0x200     add r11, 0x80
xor r15, 0x800      xor r1, 0        add r8, 0x20      or r11, 0x88
mov r12, r15        add r1, 0        add r14, r13      add r12, rbp
add r8, 0          add r1, 0        add r13, 0x80      and r12, 0x90
xor r12, 0xf0        mov r8, 0x58      add r15, rsi      or r12, 0
mov r15, 0x58        mov ax, word ptr [rdi]      add rdx, 8        add r12, 0x80
add r11, rbp       mov r8, 1        add rdx, 0x20      add r13, 0xf0
xor r15, 0x800      mov rsi, rbp      add r8, 0x20      mov r13, 0x400
and r12, 0x20        and rcx, 8        add r14, word ptr [r14]  add dword ptr [r12], eax
add r15, 0x800      sub r15, 1        xor r9, 0xfffff  and rsi, r8
mov r11, qword ptr [r11]  mov r15, rdi      or r9, 0x100000000  or r10, 8
add r15, 1          add rsi, 0x29      mov r14, word ptr [r14]  and rbx, 0x20
and r12, r9          or r15, 8        add r10, rbp      and rax, 0xfffff
add r15, 0x1          mov r15, rsi      xor r15, 0x58      xor r11, 0
mov rdx, 1          add r13, 0x80      add r9, 0x80      or rbx, r10
xor r10d, dword ptr [r8]  mov r8, rsi      add r14, rbp      add r13, r8
sub r9, r11        add r13, 0x13      add r8, r15      shl rax, 3
pushfq             xor r13b, byte ptr [rsi]  xor esi, dword ptr [r9]  add r8, rax
xor r15, 0xf0        xor r13b, 0xd2      xor r10, rbp      or rbx, r15
xor r15, 0x800      and r8, r13      add r10, 0xcc      sub r15, 0x10
and rdx, r8          or r15, 0x20      mov r14, qword ptr [r14]  or r11, r13
mov r12, rbp       sub r15, 0x90      add qword ptr [rsi], r14  mov r15, 0x5f
add r15, 0x2549b044  xor r13, 0x90      xor r11, r14      and r15, 0x3f
sub r15, 4          xor r13, 0x14      add r15, r14      xor r15, 0x12
add r11, 0x204      xor r14, rsi      xor r13, 0x12      sub r15, 0x12
or r15, 0x78        xor r14, rbp      xor r8, 0          add r8, r127
and rdx, r10        add rdx, rbp      add r14, 0x88      xor r15, 0x3f
mov rax, 0          add dword ptr [rdx], esi  xor r12, 1        xor r15, 0x3f
add r12, 0x42        xor r12, 1        add r13, 1        xor r15, 1
add r15, 0x5a        xor r13, r15      xor rdx, rbp      xor r15, rax

```

$$u64 \text{ res} = M_{13} + M_{14}$$

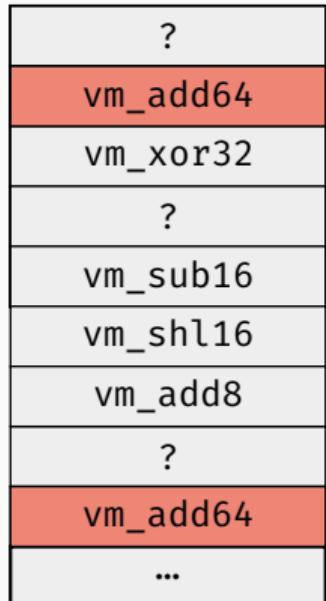
## #2: Duplicating VM Handlers



## #2: Duplicating VM Handlers

?
vm_add64
vm_xor32
?
vm_sub16
vm_shl16
vm_add8
?
vm_add64
...

## #2: Duplicating VM Handlers



## #5: Blinding VM Bytecode

```

mov    r15, 0x200          mov    r15, rdx          add   r8, 1           or    r14, r14          mov    r14, 0x200          add   r15, 0x3f
xor    r15, 0x8000         xor    r15, dword ptr [r12]    or    r8, 0x78          rax, rbp            or    rdx, 0xc0
mov    rbx, rbp          sub   r15, 0x8000         add   word ptr [rbx], r10w    and   rcx, r13          and   rdx, 0xc0
add   rdx, 0xc0          or    rdx, 0x4000        sub   r15, rax           and   rax, 4           and   r11, r14
mov    rbx, qword ptr [rbx]    nov   rsi, 0x200        sub   r8, -0x80000000     or    r15, 0x88
add   r15, 1              nov   r14, rbp           mov   r13, 0xffff         mov   rdx, qword ptr [rdx]    and   rax, 0xc0
mov    rcx, 0              sub   rsi, rsi           pop   r9             add   rdx, 0xa           or    rdi, r14
mov    r15, rbp          mov   rdi, rbp           add   rcx, rbp          and   r11, 0x78          or    si, 1
add   r15, 0xc0          nov   r8, 0x4000        add   rcx, 0x8c0        mov   r10, rbp           nov   rax, qword ptr [rax]
or    rcx, 0x88          sub   rsi, r9            add   r13, r15          cmp   r8b, 0            and   rax, 2
add   rbx, 0xb             sub   r8, rsi           add   r14, 8             mov   rdx, byte ptr [rdx]    or    rsi, 4
mov    r15, qword ptr [r15]    add   r14, 0           add   r10, 0x89         sub   r8b, 0x7fffff      or    rbi, rsi
or    r12, 0xffffffff80000000    add   rsi, rax           xor   word ptr [r10], si    mov   r8b, byte ptr [rdx]
sub   r12, 0x78             and   r8, 0x88          xor   rdx, r11           or    r11, 0x40          and   rax, 2
mov    r10, word ptr [rbx]    xor   r9, rbp           xor   r15, 1            mov   rdx, rbp           or    rsi, 4
xor   r12, r13             and   rdi, 0xfffffff80000000    sub   rdx, rbx           and   r11, 0x10          or    rbi, rsi
add   r12, 0xfffff         nov   rsi, r14           and   r10, 0x40          add   rdx, 0xc0          movzx r9, word ptr [rax]
add   r15, 0                 sub   rsi, rbp           or   rdx, 0x40           or    r14, 4            add   r8b, 0x7fffff
add   r15, 0                 sub   r13, 0x20          add   rsi, 0x5a          mov   r15, 0x12         and   r10, 0x20
add   r8, rbp              sub   r13, 0x200         add   r8, rcx           mov   rdx, qword ptr [rdx]    or    r10, 0x40
add   r12, r12              add   r8, 0x78          mov   r10, 0x88          sub   r11, r8            add   eax, dword ptr [r9]
or    r12, 0x8000           add   rsi, 4             xor   rsi, rbp           xor   r10, 0x40          xor   r10, 0x40
or    r12, r15              nov   rci, 0x200         and   rdi, 0x8000         add   rdx, 0xc0          add   eax, 0x3f50c07
add   r12, 0x8000           add   dword ptr [rsi], 0x2549b044    add   r8, 0x58          or    r11, 0x80          or    r15, 0x88
add   r11, word ptr [r15]    add   dword ptr [rsi], 0x2549b044    mov   r14, rbp           mov   r12, rbp           or    rdi, 0x90
add   r12, 0x8000           xor   rcx, 0xf0          and   rax, rdx           mov   r14, r8            or    rbi, 0x90
add   r8, 0                  add   r10, r10          add   r14, 0x20          add   r8, rbp           add   r12, 0
add   r12, 0xf0              add   rdi, 6             add   r14, 0x89         xor   r13, 4            or    rbx, 0x80
add   r11, 0x58              add   rdi, 0x80          add   rax, 0x40          pop   r10             add   rdi, 0xf0
add   r11, rbp              add   r13, 0x10          xor   rax, 0x40          qword ptr [r8], r10    mov   r13, 0x4000
add   r12, 0x8000           add   r8, 1             add   rdx, 0x78          jmp   0x4ae             add   dword ptr [r12], eax
add   r12, 0x20              add   rsi, rbp           add   rdx, 0x20          xor   rsi, 0x88         and   rsi, r8
and   r12, 0x8000           add   r9, 1             add   r14, word ptr [r14]    rbx, 0xffffffff80000000
add   r12, 0x8000           sub   rcx, 1             add   r14, 0x58          add   rsi, 0x78         or    r10, 0
add   r11, qword ptr [r11]    add   r9, rbp           add   r10, rdx           mov   r10b, 0x68         and   r13, r8
add   r12, 1                 add   r12, 0x58          xor   rax, rdx           or    r9, 0x12           and   rbx, 0x20
and   r12, r9                add   rsi, 0x29          add   r8, 0x80          or    r10, r10           add   r13, r8
add   r12, 1                 or    rcx, 8             sub   r15, 0x80          and   r15, rsi           or    rbx, 1
add   r12, r11              mov   r8, rsi           mov   r15, r13           add   r14, rbp           shl   rax, 3
add   r10d, dword ptr [r8]    add   rcx, 4             or    rcx, r12           add   r8, r15           or    r9, 8
sub   r9, r11              mov   r13b, byte ptr [rsi]    xor   esi, dword ptr [r9]    mov   r14, 0x29           add   r8, rax
pushfq r9, 0xd2             cmp   r13b, 0xd2          mov   r10, rbp           and   rdx, 0x10           xor   rbx, rdi
add   r10, 0x204             jbe   0x204          add   r10, 0xcc           xor   r14, 0x10           sub   r15, 0x10
add   r10, 0x20               add   r8, r13           sub   r15, 0x20          and   r15, 0x3f           or    r11, r13
and   rdx, r8                or    rcx, r13          xor   esi, dword ptr [r10]    or    byte ptr [r14], r10b
add   r12, rbp              or    rcx, 4             xor   r13, 0x90          mov   rax, 0x58           mov   rbx, qword ptr [r8]
add   r12, 0x20              or    rdx, rbp           add   rdi, 0x10          r8, rbp             mov   rdx, rbp
sub   r12, 4                 or    rci, 4             mov   r14, rsi           sub   r13, 0x78           sub   r13, 0x80
add   r11, 0x2549b044        sub   rcx, 0x4000        mov   rdx, rbp           add   r8, 0x127          add   rdx, 0xc0
or    r12, 0x78              add   rax, rbp           add   rdx, 0             xor   r15, rbx           add   qword ptr [rdx], 0xd
add   r12, r10              or    rcx, 0x80          add   dword ptr [rdx], esi    rdx, rbp           jmp   rbx
add   r12, 0x42              add   r10, r8000        add   r13, 0x40           r8, 0x3f            and   rsi, r8
add   r12, 0x8400             xor   rcr, 0x8000       mov   r12, 1             xor   r13, 1           or    r10, 0
add   r12, 0x5a              add   rbx, 0x5a           xor   r12, 1             mov   r10, rbp           mov   rax, rbp

```

## #5: Blinding VM Bytecode

```

r15, 0x200
xor r15, 0x800
mov rbx, rbp
add rbx, 0xc0
mov rbx, qword ptr [rbx]
mov r13, j
mov rcx, 0
mov r15, rbp
add r15, 0xc0
or rcx, 0x88
add rbx, 0xb
mov r15, qword ptr [r15]
or r12, 0xffffffff80000000
sub r12, 0x78
movzx r10, word ptr [rbx]
xor r12, r13
add r12, 0xffff
add r15, 0
mov r8, rbp
sub r12, 0x10
or r12, r12
or rcx, 0x800
movzx r11, word ptr [r15]
xor rcx, 0x800
mov r12, r15
add r8, 0
xor r12, 0xf0
mov rbx, 0x58
add r11, rbp
xor r12, 0x800
and r12, 0x20
add rbx, 0x800
mov r11, qword ptr [r11]
add rbx, 1
and r12, r9
mov rdx, 1
xor r10d, dword ptr [r8]
sub r9, r11
pushfq
xor rbx, 0xf0
xor rbx, 0x800
and rdx, r8
mov r12, rbp
xor rdx, 0x20
sub rbx, 4
add r11, 0x2549b044
or rbx, 0x78
and rdx, r10
mov rax, 0
add r12, 0x42
mov r15, rdx
xor r15, 0x800
sub r15, 0x800
or rdx, 0x800
mov r11, 0x200
mov r14, rbp
sub rsi, rsi
mov rdi, rbp
mov r8, 0x400
sub rsi, r9
sub r8, rsi
add r14, 0
add rsi, rax
and r8, 0x88
xor rsi, r14
mov rsi, rbp
add rdi, 0xc0
sub r8, rdi
add r8, 0x78
add rsi, 4
mov r15, 0x200
mov rdi, qword ptr [rdi]
add dword ptr [rsi], 0x2549
xor rcx, 0xf0
add rcs, r10
add rdi, 6
add r8, 0x400
mov ax, word ptr [rdi]
mov r8, 1
mov rsi, rbp
and rcs, 8
sub rcs, 1
mov r15, rdi
add rsi, 0x29
or rcs, 8
mov r8, rsi
add rcs, 4
mov r13b, byte ptr [rsi]
cmp r13b, 0xd2
jbe 0x204
and r8, r13
or rcs, r13
or rcs, 4
mov rbx, rbp
or rcs, 4
sub rcs, 0x400
add rax, rbp
or rcs, 0x80
add rcs, 0x80
add rbx, 0x5a

```

```

add    r8, 1          or     r14, r14
or     r8, 0x78        mov    rax, rbp
add    word ptr [rbx], r10w   and   rcx, r13
mov    r15, rax        add    rax, 4
sub    r15, rax        sub    r8, -0x80000000
pop    r9              add    r13, 0xffff
mov    rcx, rbp        and    rcx, 0x20

mov    r9, rbp
...
add    r9, 0
...
add    eax, dword ptr [r9]
...
add    eax, 0x3f505c07
...
mov    r12, rbp
...
add    r12, 0
add    dword ptr [r12], eax

add    r9, 0          add    r8, 0x80
sub    r13, 0x80        mov    r15, rsi
mov    r15, r13        add    r14, rbp
or     rcx, r12        add    r8, r15
xor    esi, dword ptr [r9]  mov    rbx, 0
mov    r10, rbp        and    rdx, 0x10
add    r10, 0xcc        mov    r14, qword ptr
sub    r15, 0x20        add    qword ptr [rsi]
xor    esi, dword ptr [r10]  pushfq
xor    r13, 0x90        xor    r11, r14
add    rdi, 0x10        add    r15, r14
mov    r14, rsi        mov    r13, 0x12
mov    rdx, rbp        mov    r8, 0
add    rdx, 0            and    r14, 0x88
add    dword ptr [rdx], esi  add    r13, 0x40
xor    r12, 1            mov    r13, 1
mov    r13, r15        mov    rdx, rbp

```

```
mov    r14, 0x200
add    rdx, 0xc0
add    r11, r14
or     r15, 0x88
mov    rdx, qword ptr [rdx]
add    rdx, 0xa
add    r11, 0x78
mov    r8b, byte ptr [rdx]
cmp    r8b, 0
je    0x4ae
si
mov    rdx, rbp
or     r11, 0x40
and    r15, 1
xor    r11, 0x10
add    rdx, 0xc0
or     r14, 4
mov    r15, 0x12
mov    rdx, qword ptr [rdx]
sub    r11, rdx
add    rdx, 4
or     r11, 0x80
mov    r8w, word ptr [rdx]
mov    r14, r8
add    r8, rbp
xor    r13, 4
pop    r10
pop    qword ptr [r8], r10
jmp    0x4ae
xor    rsi, 0x88
xor    rbx, 0xffffffffffff80000000
add    rsi, 0x78
mov    r10b, 0x68
mov    r9, 0x12
rbx, r10
and    r15, 0x78
mov    r14, rbp
or     r9, 8
add    r14, 0x29
xor    rbx, rdi
add    r15, 0x3f
or     byte ptr [r14], r10b
mov    rax, 0x58
mov    r8, rbp
sub    rsi, 0x78
add    r8, 0x127
mov    rdi, rbx
xor    rbx, 0x3f
mov    r8, qword ptr [r8]
xor    rsi, 1
mov    rax, rbp
```

```
add    r15, 0x3f
or     r15, 0xffffffff80000000
and    rsi, r9
add    rax, 0xc0
add    rdi, r14
or     rsi, 1
mov    rax, qword ptr [rax]
and    rdi, 0xfffffff
add    rax, 2
sub    rsi, 4
or     rbx, rsi
movzx  rax, word ptr [rax]

mov    r9, rbp
mov    r13, 0x200
mov    r10, 0x58
add    r9, 0
or     r10, 0x20
add    eax, dword ptr [r9]
xor    r10, 0x40
add    eax, 0x3f50c07
add    r12, 0x88
mov    r12, rbp
or     rdi, 0x90
add    r12, 0
or     rbx, 0x80
add    rdi, 0xf0
mov    r13, 0x400
add    dword ptr [r12], eax

and    rsi, 18
or     r10, 8
and    rbx, 0x20
and    rax, 0xfffff
mov    r11, 0
add    r13, r8
or     rbx, 1
shl    rax, 3
add    r8, rax
or     rbx, r15
sub    r15, 0x10
or     r11, r13
mov    rbx, qword ptr [r8]
mov    rdx, rbp
sub    r13, 0x80
add    rdx, 0xc0
add    qword ptr [rdx], 0xd
```

## #5: Blinding VM Bytecode

```

r15, 0x200          mov    r15, rdx
xor   r15, 0x800      xor    r15, dword ptr [r12]
mov   rbx, rbp        sub    r15, 0x800
add   rbx, 0xc0        or     rdx, 0x400
mov   rbx, qword ptr [rbx]    nov   rsi, 0x200
r13, 1               nov   r14, rbp
mov   rcx, 0           sub    rsi, rsi
mov   r15, rbp        nov   rdi, rbp
add   r15, 0xc0        mov    r8, 0x400
rcx, 0x88            sub    rsi, r9
add   rbx, 0xb         sub    r8, rsi
mov   r15, qword ptr [r15]    add   r14, 0
or    r12, 0xffffffff80000000    add   rsi, rax
sub   rcx, 0x78        and    r8, 0x88
movzx r10, word ptr [rbx]    xor    rsi, r14
xor   r12, r13        nov   rsi, rbp
add   r12, 0xfffff      add   rdi, 0xc0
add   r15, 0             sub    r8, rdi
mov   r8, rbp        add   r8, 0x78
sub   rcx, 0x10        add   rsi, 4
or    r12, r12        mov    rcx, 0x200
rcx, 0x800            mov   rdi, qword ptr [rdi]
movzx r11, word ptr [rcx]    add   rdi, qword ptr [rdi]
rcx, 0x800            add   rdi, qword ptr [rdi]
mov   r12, r15
xor   r12, 0xf0
mov   rbx, 0x58
add   r11, rbp
xor   rbx, 0x800
and   r12, 0x20
add   rbx, 0x800
mov   r11, qword ptr [r11]
add   rbx, 1
and   r12, r9
mov   rdx, 1
xor   r10d, dword ptr [r8]
subq  r9, r11
pushfq
xor   rbx, 0xf0
xor   rbx, 0x800
and   rdx, r8
mov   r12, rbp
xor   rdx, 0x20
sub   rbx, 4
add   r11, 0x2c549b044
or    rbx, 0x78
and   rdx, r10
mov   rax, 0
add   r12, 0x42

No influence

```

add r8, 1	or r14, r14
or r8, 0x78	mov rax, rbp
add word ptr [rbx], r10w	and rcx, r13
mov r15, rax	add rax, 4
sub r15, rax	sub r13, 0xffffffff
pop r9	add r13, 0xffff
mov rcx, rbp	and rcx, 0x20
mov r9, rbp	
...	
add r9, 0	
...	
add eax, dword ptr [r9]	
...	
add eax, 0x3f505c07	
e on underlying co	
add r12, 0	
add dword ptr [r12], eax	
add r9, 0	add r8, 0x80
sub r13, 0x80	mov r15, rsi
mov r15, r13	add r14, rbp
or rcx, r12	add r8, r15
xor esi, dword ptr [r9]	mov rbx, 0
mov r10, rbp	and rdx, 0x10
add r10, 0xcc	mov r14, qword ptr
sub r15, 0x20	qword ptr [rsi]
xor esi, dword ptr [r10]	pushfq
xor r13, 0x90	xor r11, r14
add rdi, 0x10	add r15, r14
mov r14, rsi	mov r13, 0x12
mov rdx, rbp	mov r8, 0
add rdx, 0	and r14, 0x88
add dword ptr [rdx], esi	add r13, 0x40
xor r12, 1	mov r13, 1
mov r13, r15	mov rdx, rbp

```
nov    r14, 0x200
add    rdx, 0xc0
add    r11, r14
or     r15, 0x88
nov    rdx, qword ptr [rdx]
add    rdx, 0xa
add    r11, 0x78
nov    r8b, byte ptr [rdx]
rep    r8b, 0
je     r14, 0x40
si
nov    rdx, rbp
or     r11, 0x40
and    r15, 1
xor    r11, 0x10
add    rdx, 0xc0
or     r14, 4
nov    r15, 0x12
nov    rdx, qword ptr [rdx]
sub    r11, r8
rdx, 4
or     r11, 0x80
mov    r8w, word ptr [rdx]
rsi]
ode's semantic equivalence between the two assembly snippets.
```

```
add    r15, 0x3f
or     rsi, 0xffffffff80000000
and    rsi, r9
add    rax, 0x0c
add    rdi, r14
or     rsi, 1
mov    rax, qword ptr [rax]
and    rdi, 0x7fffffff
add    rax, 2
sub    rsi, 4
or     rbx, rsi
movzx rax, word ptr [rax]
mov    r9, rbp
mov    r13, 0x200
mov    r10, 0x58
add    r9, 0
or     r10, 0x20
add    eax, dword ptr [r9]
xor    r10, 0x40
add    eax, 0x3F505c07
add    r15, 0x88
mov    r12, rbp
i      i, 0x90
i      2, 0
x      0x80
i      0xF0
i      3, 0x400
ordptr[r12], eax
00 and    r15, r8
or     r10, 0x8
and    rbx, 0x20
and    rax, 0xfffff
mov    r11, 0
add    r13, r8
or     rbx, 1
shl    rax, 3
add    r8, rax
or     rbx, r15
sub    r15, 0x10
or     r11, r13
mov    rbx, qword ptr [r8]
mov    rdx, rbp
sub    r13, 0x80
add    rdx, 0xc0
add    qword ptr [rdx], 0xd
jmp    rbx
```

No influence on underlying code's semantics

# #3: No Central VM Dispatcher

```

mov    r15, 0x200          mov    r15, rdx          add   r8, 1           or    r14, r14          mov    r14, 0x200          add   r15, 0x3f
xor    r15, 0x800          xor    r10d, dword ptr [r12]  or    r8, 0x78          mov    rax, rbp          add   rdx, 0xc0
mov    rbx, rbp          sub   r15, 0x800          add   word ptr [rbx], r10w  and   r15, r13          add   r11, r14
add   rbx, 0xc0          or    rdx, 0x400          mov    r15, rax          add   rax, 4           or    r15, 0x88
mov    rbx, qword ptr [rbx]  nov   r15, 0x200          sub   r8, 0x80000000  mov    rdx, qword ptr [rdx]
nov   r13, 1             nov   r14, rbp          pop   r9           or    r14, r14          add   r15, 0x3f
mov    r13, 0             sub   r15, rax          add   rcx, 0xffff          mov    r15, 0x80000000
nov   r15, 0             sub   r15, rax          add   rcx, rbp          add   rdx, 0x80
and   r15, 0             sub   r15, rax          and   r15, r13          add   r11, r14
add   r15, 0x800          mov   r15, rax          add   r15, 0x88          and   rsi, r9
or    r15, 0x800          mov   r15, rax          mov    r10, rbp          add   rax, 0xc0
add   r15, 0x800          sub   r15, rax          add   rcx, 0xc0          and   rdi, r14
or    r15, 0x800          sub   r15, rax          add   rcx, qword ptr [rcx]  mov    r15, 0x80000000
add   r15, 0x800          sub   r15, rax          add   r13, 0x15          cmp   rbb, byte ptr [rdx]
or    r15, 0x800          sub   r15, rax          add   r13, r15          je    rbb, 0
add   r15, 0x800          sub   r15, rax          add   r14, r8           sub   rsi, 4
or    r15, 0x800          sub   r15, rax          add   r14, r8           or    r15, 0x3f
add   r15, 0x800          sub   r15, rax          mov    r10, 0x89           or    r15, 0x80000000
or    r15, 0x80000000      add   r15, 0x800          xor   word ptr [r10], si  add   r15, 0x3f
add   r15, 0x80000000      add   r15, 0x800          xor   rdx, 0x11           or    r15, 0x80000000
sub   r15, 0x80000000      add   r15, 0x800          xor   r10d, dword ptr [r9]  and   r15, r14
and   r15, 0x80000000      add   r15, 0x800          mov   r15, 0x80000000  xor   r15, 0x80000000
movzx r15, word ptr [rbx]  xor   r15, 0x80000000  add   rdx, 0xc0          add   r13, 0x200
movzx r15, word ptr [rbx]  xor   r15, 0x80000000  and   r15, 0x80000000  mov   r10, 0x58
add   r15, 0x80000000      xor   r15, 0x80000000  sub   r13, 0xf0          add   r10, 0x20
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  or    r14, 1
add   r15, 0x80000000      xor   r15, 0x80000000  sub   r13, 0x20          or    r14, 1
add   r15, 0x80000000      xor   r15, 0x80000000  add   r15, 0x5a          mov   r15, 0x80000000
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  sub   r11, r8
add   r15, 0x80000000      xor   r15, 0x80000000  movzx r15, word ptr [rsi]  add   rdx, 4
add   r15, 0x80000000      xor   r15, 0x80000000  xor   r15, 0x80000000  or    r11, 0x800
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  mov   r15, 0x80000000
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  and   r15, r14
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  xor   r15, 0x80000000
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   r13, 0x200
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  mov   r10, 0x58
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   r10, 0x20
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   rax, 0xc0
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  xor   r10, 0x40
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   eax, dword ptr [r9]
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  xor   r10, 0x40
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   eax, 0x3f05c057
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  xor   r15, 0x80000000
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   r15, 0x80000000
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  xor   r12, rbp
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  or    rdi, 0x90
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   r12, 0
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  or    rbx, 0x80
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   rdi, 0xf0
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  mov   r13, 0x400
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   dword ptr [r12], eax
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  and   rsi, r8
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  or    r10, 0
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  and   rbx, 0x20
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  and   rax, 0xffff
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  mov   r11, 0
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   r13, r8
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  or    rbx, 1
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  shl   rax, 3
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   r8, rax
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  or    rbx, r15
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  sub   r15, 0x10
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  or    r11, r13
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  mov   r15, 0x80000000
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  sub   r13, 0x80
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   rdx, 0xc0
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  add   qword ptr [rdx], 0xd
add   r15, 0x80000000      xor   r15, 0x80000000  mov   r15, 0x80000000  jmp   rbx
pushfq
xor    r15, 0xf8          jbe   0x204          add   r10, rbp          and   r14, qword ptr [r14]
xor    r15, 0x800          and   r8, r13          mov   r15, 0x800          mov   r15, qword ptr [r14]
and   rdx, r8             or    r15, 0x20          xor   r15, 0x800          and   r15, 0x3f
mov    r12, rbp          or    r15, 0x20          mov   r15, 0x800          mov   r15, 0x80000000
xor    r15, 0x800          xor   r15, 0x90          xor   r15, 0x800          add   r15, 0x3f
and   rdx, r10          add   r15, 0x800          pushfq            or    r15, 0x80000000
mov    rax, 0              add   r15, 0x800          xor   r11, r14          mov   r15, 0x80000000
add   r12, 0x42          add   r15, 0x800          add   r15, 0x800          sub   rsi, 0x78
add   r15, 0x80000000      xor   r15, 0x800          mov   r15, 0x800          add   r8, 0
add   r15, 0x80000000      xor   r15, 0x800          mov   r15, 0x800          and   r14, 0x88
add   r15, 0x80000000      xor   r15, 0x800          xor   r12, 1           xor   r15, 0x3f
add   r15, 0x80000000      xor   r15, 0x800          add   r13, 1           xor   rsi, 1
add   r15, 0x80000000      xor   r15, 0x800          mov   r15, 0x800          mov   r15, 0x80000000
add   r15, 0x80000000      xor   r15, 0x800          mov   r15, 0x800          add   rdx, rbp
add   r15, 0x80000000      xor   r15, 0x800          mov   r15, 0x800          add   qword ptr [rdx], 0xd
add   r15, 0x80000000      xor   r15, 0x800          mov   r15, 0x800          jmp   rax, rbp

```

### #3: No Central VM Dispatcher

```

r15, 0x800    mov    r15, rdx
xor r15, 0x800 xor    r15, dword ptr [r12]
mov rbx, rbp sub   r15, 0x800
add rbx, 0xc0 or    rdx, 0x400
mov rbx, qword ptr [rbx] mov   rsi, 0x200
mov r13, 1 mov   r14, rbp
mov rcx, 0 sub   rsi, rsi
mov r15, rbp mov   rdi, rbp
add r15, 0xc0 nov  r8, 0x400
or rcx, 0x88 sub   rsi, r9
add rbx, 0xb sub   r8, rsi
mov r15, qword ptr [r15] add   r14, 0
or r12, 0xffffffff80000000 add   rsi, rax
sub r12, 0x78 and   r8, 0x88
mov r10, word ptr [rbx] xor   rsi, r14
xor r12, r13 mov   rsi, rbp
add r12, 0xfffff add   rdi, 0xc0
add r15, 0 sub   r8, rdi
mov r8, rbp add   r8, 0x78
sub rcx, 0x10 add   rsi, 4
or r12, r12 mov   rcx, 0x200
or rcx, 0x800 mov   rdi, qword ptr [rdi]
mov r11, word ptr [r15] add   dword ptr [rsi], 0x2549
xor rcx, 0x800 xor   rcx, 0xf0
mov r12, r15 add   rdx, r10
add r8, 0 add   rdi, 6
xor r12, 0xf0 mov   r8, 0x400
mov rbx, 0x58 mov   ax, word ptr [rdi]
add r11, rbp mov   r8, 1
xor rbx, 0x800 mov   rst, rbp
and r12, 0x20 and   rcx, 8
add rbx, 0x800 sub   rcx, 1
mov r11, qword ptr [r11] mov   rcx, rdi
add rbx, 1 add   rsi, 0x209
and r12, r9 or    rcx, 8
mov rdx, 1 mov   r8, rsi
xor r10d, dword ptr [r8] add   rcx, 4
sub r9, r11 mov   r13b, byte ptr [rsi]
pushfq xor   rbs, 0xf0 cmp   r13b, 0xd2
xor rbs, 0x800 jbe  0x204
and rdx, r8 and   r8, r13
mov r12, rbp or    rcx, r13
xor rdx, 0x20 or    rcx, 4
sub rbx, 4 mov   rbx, rbp
add r11, 0x2549b044 or    rcx, 4
or   rbs, 0x78 sub   rcx, 0x400
and rdx, r10 add   rax, rbp
mov rax, 0 or    rcx, 0x88
add r12, 0x42 add   rcx, 0x5a

```

```

add    r8, 1          or     r14, r14
or     r8, 0x78        mov    rax, rbp
add    word ptr [rbx], r10w   and   rcx, r13
mov    r15, rax        add    rax, 4
sub    r15, rax        sub    r8, -0x8000
pop    r9              add    r13, 0xffffffff
mov    rcx, rbp        and    rcx, 0x20

or     rbx, 1
shl    rax, 3
add    r8, rax
or     rbx, r15
sub    r15, 0x10
or     r11, r13
mov    rbx, qword ptr [r8]
mov    rdx, rbp
sub    r13, 0x80
add    rdx, 0xc0
add    qword ptr [rdx], 0xd
jmp    rbx

add    r9, 0           add    r8, 0x80
sub    r13, 0x80        mov    r15, rsi
mov    r15, r13        add    r14, rbp
or     rcx, r12        add    r8, r15
xor    esi, dword ptr [r9]   mov    rbx, 0
mov    r10, rbp        and    rdx, 0x10
add    r10, 0xcc        mov    r14, qword
sub    r15, 0x20        add    qword ptr [
xor    esi, dword ptr [r10]   pushfq
xor    r13, 0x90        xor    r11, r14
add    rdi, 0x10        add    r13, r14
mov    r14, rsi        mov    r13, 0x12
mov    rdx, rbp        mov    r8, 0
add    rdx, 0            and    r14, 0x88
add    dword ptr [rdx], esi   add    r13, 0x40
xor    r12, 1            mov    r13, 1
mov    r13, r15        mov    rdx, rbp

```

The diagram illustrates a control flow graph (CFG) for the provided assembly code. Nodes represent assembly instructions, and edges represent control flow between them. The nodes are arranged vertically, and edges are shown as diagonal lines connecting the exit of one node to the entry of another.

- Top Node:** mov r14, 0x2000
- Second Node:** add rdx, 0xc0
- Third Node:** add r11, r14
- Fourth Node:** or r15, 0x88
- Fifth Node:** mov rdx, qword ptr [rdx]
- Sixth Node:** add rdx, 0xa
- Seventh Node:** add r11, 0x78
- Eighth Node:** mov r8b, byte ptr [rdx]
- Ninth Node:** cmp r8b, 8
- Tenth Node:** je 0x49e
- Eleventh Node:** mov rdx, rbp
- Twelfth Node:** or r11, 0x40
- Thirteenth Node:** and r15, 1
- Fourteenth Node:** xor r11, 0x10
- Fifteenth Node:** add rdx, 0xc0
- Sixteenth Node:** or r14, 4
- Seventeenth Node:** mov r15, 0x12
- Eighteenth Node:** mov rdx, qword ptr [rdx]
- Nineteenth Node:** sub r11, r8
- Twenty-First Node:** add r11, r8
- Twenty-Second Node:** or r11, 0x80
- Twenty-Third Node:** r11, 0x80
- Twenty-Fourth Node:** mov r8w, word ptr [rdx]
- Twenty-Fifth Node:** mov r14, 8
- Twenty-Sixth Node:** add r8, rbp
- Twenty-Seventh Node:** xor r13, 4
- Twenty-Eighth Node:** pop r10
- Twenty-Ninth Node:** mov qword ptr [r10], r10
- Thirty-Second Node:** jmp 0x4ae
- Thirty-Third Node:** xor rsi, 0x88
- Thirty-Fourth Node:** xor rbx, 0xffffffffffff00000000
- Thirty-Fifth Node:** add rsi, 0x78
- Thirty-Sixth Node:** xor r10b, 0x68
- Thirty-Seventh Node:** r9, 0x12
- Thirty-Eighth Node:** rbx, r10
- Thirty-Ninth Node:** or r15, 0x78
- Forty-Second Node:** xor r14, rbp
- Forty-Third Node:** r9, 8
- Forty-Fourth Node:** add r14, 0x29
- Forty-Fifth Node:** xor rbx, rdi
- Forty-Sixth Node:** r15, 0x3f
- Forty-Seventh Node:** byte ptr [r14], r10b
- Forty-Eighth Node:** mov rax, 0x5b
- Forty-Ninth Node:** mov r8, rbp
- Fiftieth Node:** sub rsi, 0x78
- Fifti-First Node:** add r8, 0x127
- Fifti-Second Node:** mov rdi, rbx
- Fifti-Third Node:** xor rbx, 0x3f
- Fifti-Fourth Node:** mov r8, qword ptr [r8]
- Fifti-Fifth Node:** xor rsi, 1
- Fifti-Fifth Node:** mov rax, rbp

```
add    r15, 0x3f
or     rsi, 0xffffffffffff00000000
and    rsi, r9
add    rax, 0xc0
add    rdi, r14
or     rsi, 1
mov    rax, qword ptr [rax]
and    rdi, 0xffffffff
add    rax, 2
sub    rsi, 4
or     rbx, rsi
movzx rax, word ptr [rax]
mov    r9, rbp
mov    r13, 0x200
mov    r10, 0x58
add    r9, 0
or     r10, 0x20
add    eax, dword ptr [r9]
xor    r10, 0x40
add    eax, 0x3f505c07
add    r15, 0x88
mov    r12, rbp
or     rdi, 0x90
add    r12, 0
or     rbx, 0x80
add    rdi, 0xf0
mov    r13, 0x400
add    dword ptr [r12], eax
and    rsi, r8
or     r10, 8
and    rbx, 0x20
and    rax, 0xfffff
mov    r11, 0
add    r13, r8
or     rbx, 1
shl    rax, 3
add    r8, rax
or     rbx, r15
sub    r15, 0x10
or     r11, r13
mov    rbx, qword ptr [r8]
mov    rdx, rbp
sub    r13, 0x80
add    rdx, 0xc0
add    qword ptr [rdx], 0xd
jmp    rbx
```

# #3: No Central VM Dispatcher

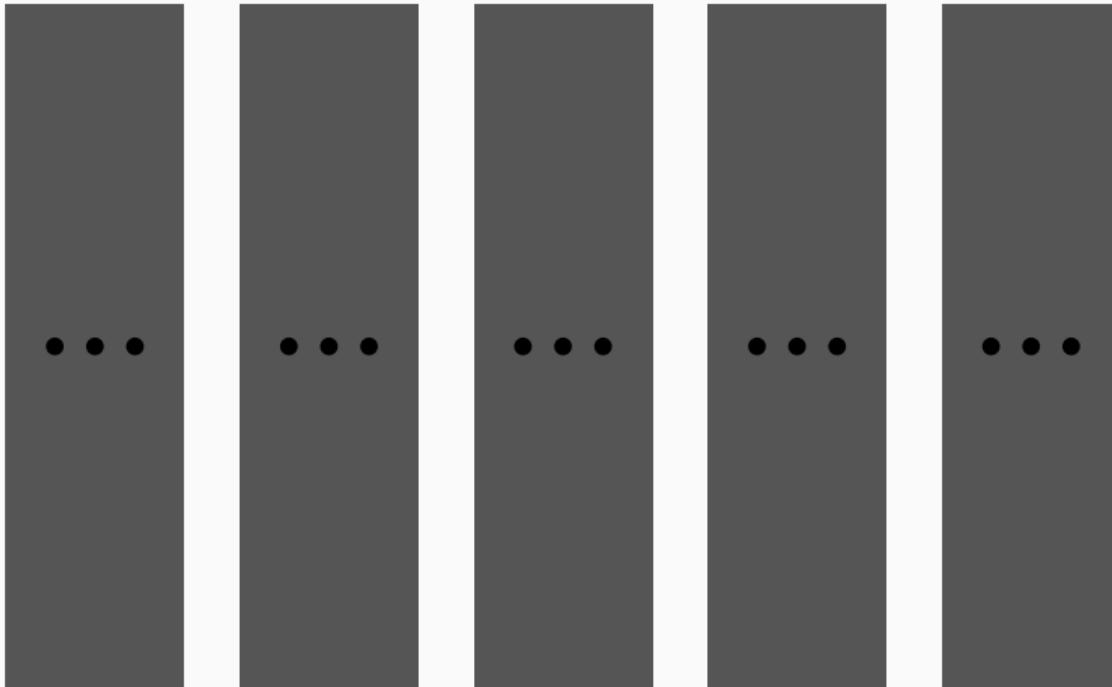
```

mov r15, 0x200          mov r15, rdx          add r8, 1           or r14, r14        mov r14, 0x200        add r15, 0x3f
xor r15, 0x800          xor r10d, dword ptr [r12]    or r8, 0x78        mov rax, rbp        rdx, 0xc0
mov rbx, rbp            sub r15, 0x800         add word ptr [rbx], r10w   and rcx, r13      r11, r14
add rbx, 0xc0            or rdx, 0x400         mov r15, rax        add rax, 4       or r15, 0x88
mov r15, qword ptr [rbx] nov r15, 0x200        sub r15, rax        mov rdx, qword ptr [rdx]
add r15, 1               mov r14, rbp        sub r8, 0x80000000  and r15, 0xffff
mov r15, 0                sub rsi, rsi        and rcx, 0x20      mov r8b, byte ptr [rdx]
add r15, rbp             mov rdi, rbp        cmp r8b, 0         or r15, 0x3f
or rcx, 0x88             sub r15, rbp        je 0x49fe         or r15, 0x7fffff
add rbx, 0xb              sub rsi, r9         mov rdx, rbp        and rsi, 1
mov r15, qword ptr [r15] add r15, 0             add r11, 0x40        or rax, 0xc0
or r12, 0xffffffff000000 add r15, rax        xor r11, 0x40      and rdt, r14
sub r15, 0x78             and r8, 0x88        add r15, 0x10        or rsi, 1
movzx r15, word ptr [rbx] xor r15, r14        xor r11, 0x10      mov rax, 0x200
add r12, 0xffff           add rsi, rbp        add rdx, 0xa        and rsi, 1
add r15, 0x8              sub r15, rdx        or r14, 4         or rsi, 4
add r15, 0x200            add r8, rdi        mov r15, 0x12        mov rax, word ptr [rax]
sub r15, 0x10              sub rsi, 4         sub r11, r8        mov r9, rbp
add r15, 0x200            add r15, 0x10        mov r15, 0x12        mov r13, 0x200
sub r15, 0x12              add rsi, 4         sub r11, r8        mov r10, 0x58
add r15, 0x200            add r15, r13        mov r15, 0x12        or r9, 0
sub r15, 0x12              add r11, r13        sub r11, r8        add r10, 0x20
add r15, 0x200            mov r15, qword ptr [r8]  add r11, 0x4        add eax, dword ptr [r9]
sub r15, 0x12              mov rdt, qword ptr [rdi]  or r11, 0x8        xor r10, 0x40
add r15, 0x200            add r15, 0xd        add r8b, word ptr [rdx]  add eax, 0x3f05c07
sub r15, 0x12              add rsi, 0xd        mov r8b, word ptr [rdx]  add r15, 0x88
add r15, 0x200            add r15, 0xd        add r15, 0x3f        add r12, rbp
sub r15, 0x12              add rsi, 0xd        or r15, 0x7fffff  or rdi, 0x98
add r15, 0x200            add r15, 0xd        and r11, 0         add r12, 0
sub r15, 0x12              add rsi, 0xd        or rbx, 0x20        or rbx, 0x80
add r15, 0x200            add r15, 0xd        and rax, 0xffff  add rdi, 0xf8
sub r15, 0x12              add rsi, 0xd        mov r11, 0         add r13, 0x400
add r15, 0x200            add r15, 0xd        add dword ptr [r12], eax
sub r15, 0x12              add rsi, 0xd        and rsi, r8        add r13, 0x8
add r15, 0x200            add r15, 0xd        or r10, 0         and rbx, 0x20
sub r15, 0x12              add rsi, 0xd        and rax, 0xffff  and rax, 0x10
add r15, 0x200            add r15, 0xd        mov r11, 0         add r11, 0
sub r15, 0x12              add rsi, 0xd        add r15, 0x78        add r13, 0x8
add r15, 0x200            add r15, 0xd        or r15, 0x8        add r12, rax
sub r15, 0x12              add rsi, 0xd        and r15, 0x3f        add r15, r15
add r15, 0x200            add r15, 0xd        or r15, 0x7fffff  sub r15, 0x10
sub r15, 0x12              add rsi, 0xd        and r11, r13        or r11, r13
add r15, 0x200            add r15, 0xd        or rbx, qword ptr [r8]  mov rdx, rbp
sub r15, 0x12              add rsi, 0xd        and r13, 0x88        sub r13, 0x88
add r15, 0x200            add r15, 0xd        add r13, 0x40        add rdx, 0xc0
sub r15, 0x12              add rsi, 0xd        xor r12, 1         add qword ptr [rdx], 0xd
add r15, 0x200            add r15, 0xd        add r13, 1         jmp r8b
sub r15, 0x12              add rsi, 0xd        xor rsi, 1         rax, rbp
add r15, 0x200            add r15, 0xd        mov rdx, rbp

```

**Split at indirect control-flow transfers**

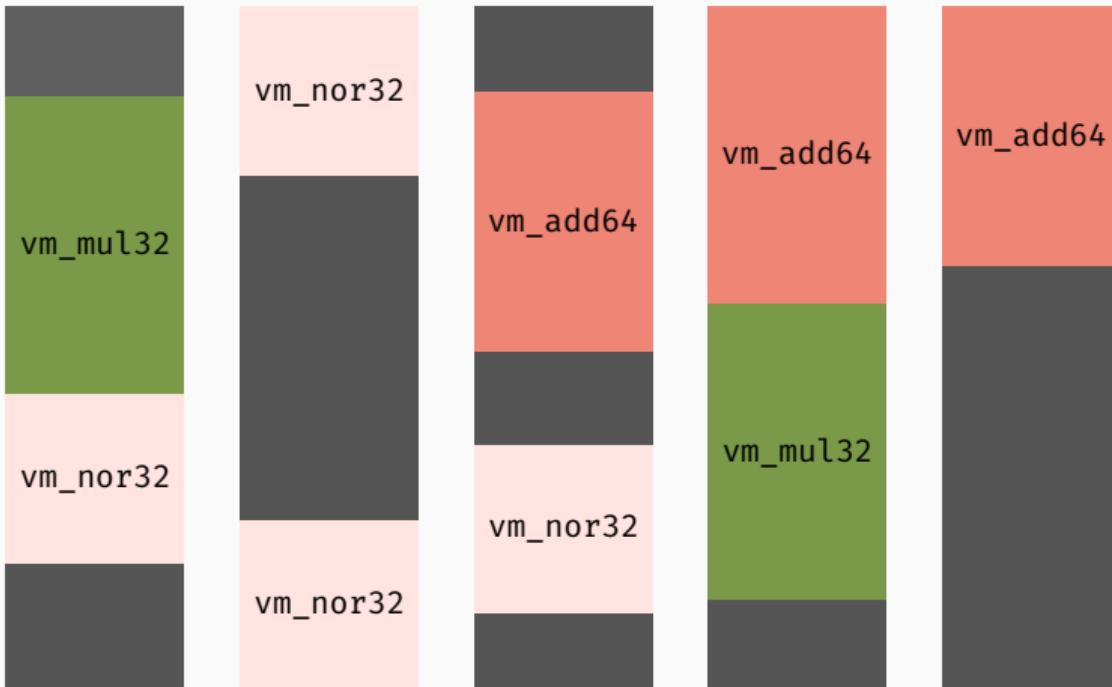
## #4: No Explicit Handler Table



## #4: No Explicit Handler Table



## #4: No Explicit Handler Table



# Conclusion

1. syntactic complexity insignificant

1. syntactic complexity insignificant
2. semantic complexity low within specified boundaries

1. syntactic complexity insignificant
2. semantic complexity low within specified boundaries
3. learn underlying code's semantics despite obfuscation

1. syntactic complexity insignificant
2. semantic complexity low within specified boundaries
3. learn underlying code's semantics despite obfuscation

Program Synthesis as an orthogonal approach to traditional techniques

# Limitations

choosing *meaningful* code window boundaries

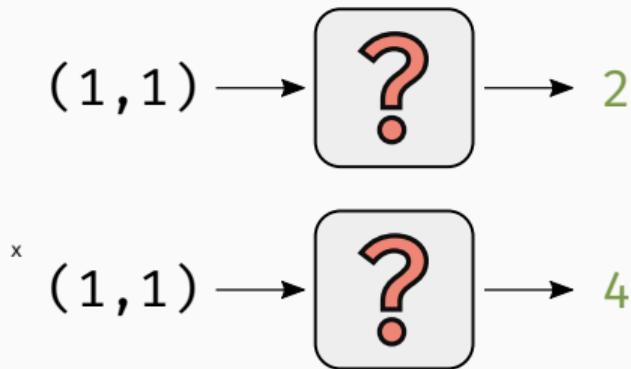
$$(x \oplus y) + 2 \cdot (x \wedge y) \quad \text{vs.} \quad (x \oplus y) + 2$$

constants

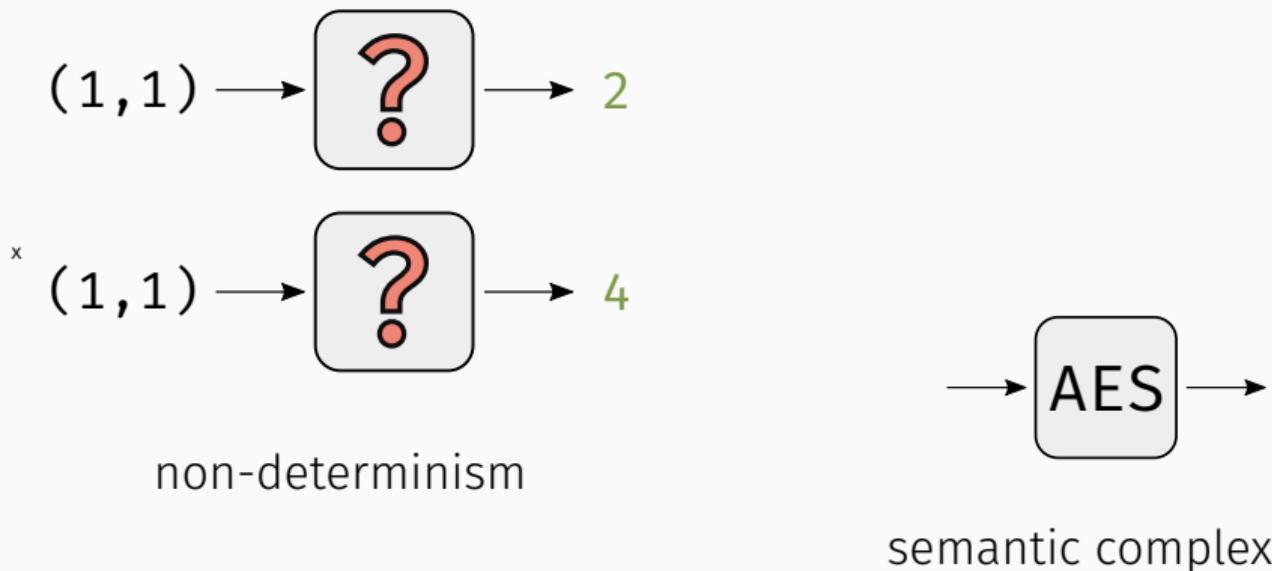
$$x + 15324326921$$

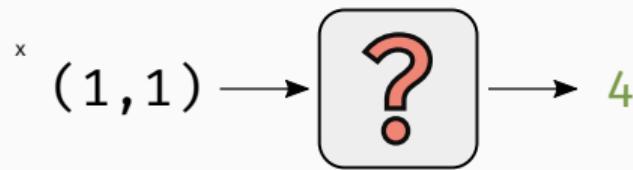
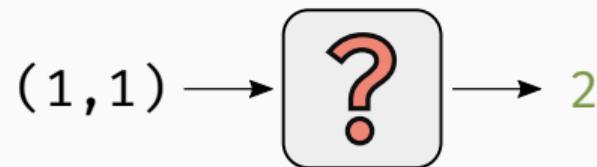
control-flow operations

$$x \ ? \ y \ : \ z$$

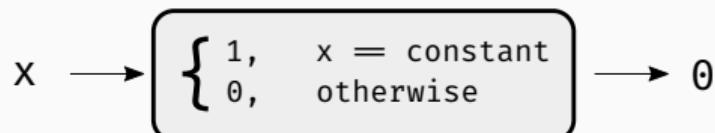


non-determinism





non-determinism



point functions



semantic complexity

# Do try it at home!

Code Issues 1 Pull requests 0 Projects 0 Insights

Branch: master **syntia / samples /** Create new file Find file History

 mrphrazer	added MBA samples from tigress	Latest commit 91a5c16 7 days ago
..		
 info	added VM handler samples for vmprotect and themida	7 days ago
 mba/tigress	added MBA samples from tigress	7 days ago
 themida/tiger_white	added VM handler samples for vmprotect and themida	7 days ago
 vmprotect	added VM handler samples for vmprotect and themida	7 days ago
 tigress_mba_trace.bin	initial commit	15 days ago
 vmprotect_add16_trace.bin	initial commit	15 days ago

- obfuscation techniques (opaque predicates, VM, MBA)
- symbolic execution for syntactic deobfuscation
- program synthesis for semantic deobfuscation

<https://github.com/RUB-SysSec/syntia>