# Pharo Virtual Machine News from the Front

P. Tesone - G. Polito - ESUG'22 @tesonep pablo.tesone@inria.fr @guillep guillermo.polito@univ-lille.fr











#### 2022 VM+ Team



#### **Virtual Machine Execution Engine**





#### **ARM64 Backend**

- ARM64 is now pervasive:
  - New Apple M1
  - Raspberry Pi 4
  - Microsoft Surface Pro X
  - PineBook Pro

move r1 #1
move r2 #17
checkSmallInt
checkSmallInt
add r3 r1 r2
checkSmallInt
move r1 r3
ret

JIT compiler IR





#### **Working Directly on Real Hardware**

- How to do a **partial** implementation, in an iterative way?
- Hardware availability: did not have access to an Apple M1 yet
- Slow Change-Compile-Test cycle
- **Bug reproduction** is a demanding task



### **Simulation Environment**





#### **Extending Simulation with Unit Tests**

Simulation Environr	nent (Pharo) VM	
Testing	Interpreter GC JIT Compiler	Transpiled to
infrastructure	Heap Native Code Cache	
		Production VM (C)
Unicorn	LLVM Disassembler	



testPushConstantZeroBytecodePushesASmallIntegerZero

self compile: [ compiler genPushConstantZeroBytecode ].
self runGeneratedCode.

self assert: self popAddress equals: (memory integerObjectOf: 0)



Reusable test fixtures covering e.g.,

- trampoline and stub compilation
- heap initialization

testPushConstantZeroBytecodePushesASmallIntegerZero

self compile: [ compiler genPushConstantZeroBytecode ].
self runGeneratedCode.

self assert: self popAddress equals: (memory integerObjectOf: 0)



Reusable test fixtures covering e.g.,

- trampoline and stub compilation
- heap initialization

testPushConstantZeroBytecodePushesASmallIntegerZero

Compiler internal DSL

self compile: [ compiler genPushConstantZeroBytecode
self runGeneratedCode.

self assert: self popAddress equals: (memory integerObjectOf: 0)



Reusable test fixtures covering e.g.,

- trampoline and stub compilation

- heap initialization

testPushConstantZeroBytecodePushesASmallIntegerZero

Compiler internal DSL

self compile: [ compiler genPushConstantZeroBytecode
self runGeneratedCode.

self assert: self popAddress equals: (memory integerObjectOf: 0)

JIT Execution helpers such as e.g., - run all code between two addresses

- run until the PC hits an address



#### **Blackbox testing**

testPushConstantZeroBytecodePushesASmallIntegerZero

self compile: [ compiler genPushConstantZeroBytecode ].
self runGeneratedCode.

self assert: self popAddress equals: (memory integerObjectOf: 0)

Depend only on observable behaviour Reusable on different backends / Resistant to changes in the implementation



### **Cross-Compilation, Cross-Execution**



#### testPushConstantZeroBytecodePushesASmallIntegerZero

- self compile: [ compiler genPushConstantZeroBytecode ].
- self runGeneratedCode.
- self assert: self popAddress equals: (memory integerObjectOf: 0)



#### There is no silver bullet

- Simulators are cheap, but not 100% trustworthy
- Full execution (simulated or on real HW)
  - more expensive to run
  - cannot unit-test it (less controllable)
- Unit tests only exercise specific scenarios
- Full executions exercise not yet covered scenarios



### **Our testing Workflow**

- Simulate the execution, less than you run tests
- Run the real app, less than you simulate
- Go back and forth:
  - Turn full execution failures into tests
  - Fix with the aid of the test:
     => unit test are faster to run
     => easier to debug
     => detect regressions





### **Testing & TDDing the VM**

- No useful unit tests by ~06/2020
- Large manual testing effort during 2020 while porting to ARM64bits
  - Extended VM simulation with a (TDD compatible) unit testing infrastructure
  - **450+** written tests on the interpreter and the garbage collector\*
  - **580+** written tests on the JIT compiler\*
  - Parametrisable for 32 and 64bits, ARM32, ARM64, x86, x86-64



### **Testing & TDDing the VM**

#### 1040+ tests, are they enough?

- No useful unit tests by ~06/2020
- Large manual testing effort during 2020 while porting to ARM64bits
  - Extended VM simulation with a (TDD compatible) unit testing infrastructure
  - 450+ written tests on the interpreter and the garbage collector\*
  - **580+** written tests on the JIT compiler\*
  - Parametrisable for 32 and 64bits, ARM32, ARM64, x86, x86-64



#### How can we automatically test VMs?



#### **Challenges of VM Test Generation**









• How do we determine what is the *expected output* of a generated test?





















#### Interpreter are Executable Semantics Pharo VM Example

- 1 Interpreter >> bytecodePrimAdd
- 2 | rcvr arg result |
- <sup>3</sup> rcvr := self internalStackValue: 1.
- 4 arg := self internalStackValue: 0.
- 5 (objectMemory areIntegers: rcvr and: arg) if True: [
- 7 "Check for overflow"
- 8 (objectMemory isIntegerValue: result) if True: [
- 9 self

10

11

12

- internalPop: 2
- thenPush: (objectMemory integerObjectOf: result).
  - ^ self fetchNextBytecode "success"]].
- 13 "Slow path, message send"
- 14 self normalSend



#### Interpreter are Executable Semantics Pharo VM Example









#### Interpreter VS Compiled Code Pharo VM Example

1	Interpreter >> bytecodePrimAdd
2	rcvr arg result
3	rcvr := self internalStackValue: 1.
4	arg := self internalStackValue: 0.
5	(objectMemory areIntegers: rcvr and: arg) if True: [
6	result := (objectMemory integerValueOf: rcvr) + (
	objectMemory integerValueOf: arg).
7	"Check for overflow"
8	(objectMemory isIntegerValue: result) if True: [
9	self
10	internalPop: 2
11	thenPush: (objectMemory integerObjectOf: result).
12	^ self fetchNextBytecode "success"]].
13	"Slow path, message send"

14 self normalSend

1	# previous bytecode IR
2	checkSmallInteger t0
3	jumpzero notsmi
4	checkSmallInteger t1
5	jumpzero notsmi
6	t2 := t0 + t1
7	jumplfNotOverflow continue
8	notsmi: #slow case first send
9	t2 := send #+ t0 t1
10	continue:
11	# following bytecode IR



#### Interpreter VS Compiled Code Pharo VM Example





#### **Concolic Testing through Meta-interpretation**

Idea: Guide test generation by looking at the implementation





- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

x	У	constraints	next?

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

X	У	constraints	next?
0	0	x <= 100	



- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

x	У	constraints	next?
0	0	x <= 100	x > 100



- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

x	У	constraints	next?
0	0	x <= 100	x > 100
101	0		



- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

X	У	constraints	next?
0	0	x <= 100	x > 100
101	0	x > 100, y != 1023	



- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

x	У	constraints	next?
0	0	x <= 100	x > 100
101	0	x > 100, y != 1023	x > 100, y == 1023


## **Concolic Testing by Example**

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

x	У	constraints	next?
0	0	x <= 100	x > 100
101	0	x > 100, y != 1023	x > 100, y == 1023
101	1023		

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05 Set et al. CUTE: a concolic unit testing engine for C. FSE'05



## **Concolic Testing by Example**

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover all execution paths

x	У	constraints	next?
0	0	x <= 100	x > 100
101	0	x > 100, y != 1023	x > 100, y == 1023
101	1023	x > 100, y != 1023	finished!

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05 Set et al. CUTE: a concolic unit testing engine for C. FSE'05



## **Some Numbers**

- 3 bytecode compilers + 1 native method compiler
- 4928 tests generated
- 478 differences

Compiler	# Tested Instructions	# Interpreter Paths	# Curated Paths	# Differences (%)
Native Methods (primitives)	112	2024	1520	440 (28,95%)
Simple Stack BC Compiler	175	1308	1136	18 (1,59%)
Stack-to-Register BC Compiler	175	1308	1136	10 (0,88%)
Linear-Scan Allocator BC Compiler	175	1308	1136	10 (0,88%)
Total	637	5948	4928	478 (9,7%)



### **Analysis of Differences through Manual Inspection**

- 91 causes, 6 different categories
- Errors both in the interpreter AND the compilers
- 14 causes of *segmentation faults*!

Family	# Cases
Missing interpreter type check	1
Missing compiled type check	13
Optimisation difference	10
Behavioral difference	5
Missing Functionality	60
Simulation Error	2



## **Practical and Cheap**

- Test generation ~5 minutes
- Total run time of ~10 seconds
  Avg 30ms per instruction





## More in the PLDI article!

- **Discovered Bugs**
- **Concolic Model**
- Testing Infrastructure

**PLDI'22** 

### **Interpreter-Guided Differential JIT Compiler Unit** Testing

Guillermo Polito Univ. Lille, CNRS, Inria, Centrale Lille, Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille France guillermo.polito@univ-lille.fr

Stéphane Ducasse UMR 9189 CRIStAL France stephane.ducasse@inria.fr

Pablo Tesone Pharo Consortium Univ. Lille, Inria, CNRS, Centrale Lill UMR 9189 CRIStAL France pablo.tesone@inria.fr

### Abstract

Modern language implementations using Virtual Machines feature diverse execution engines such as byte-code interpreters and machine-code dynamic translators, a.k.a. JIT compilers. Validating such engines requires not only validating each in isolation, but also that they are functionally equivalent. Tests should be duplicated for each execution engine, exercising the same execution paths on each of them.

In this paper, we present a novel automated testing approach for virtual machines featuring byte-code interpreters. Own collection wood concelling moto intermediation

San Diego, CA, USA. ACM, New York, NY, USA, 12 pages. //doi.org/10.1145/3519939.3523457

### **1** Introduction

Modern Virtual Machines support code generation fo compilation and dynamic code patching for techniques as inline caching. They are often structured around a code interpreter, a baseline JIT compiler, and a specula inliner. This complexity is aggravated when the VM by and runs on multiple target architectures [1]. Validating



# Is that all?

## **Ongoing RISCV64 Port**

- Currently under development: Real HW testing stage
- Taking advantage of our harness test suite
- Improving tests and scenarios

- Collaboration with Q. Ducasse, P. Cortret, L. Lagadec from ENSTA Bretagne
- Future work on: *Hardware-based security enforcement*





### **Single Instruction Multiple Data Extensions**





## **SIMD Design Space**

- VM Primitives
  - Specialised
  - Faster, less checks
- Vectorised Bytecode
  - Composable
  - Safe at the expense of speed





## **Tools for Debugging**

- Machine Code Debugger
- Compiler IR Visualisations
- Disassembler DSL

× - 🗆												
IR Instructions	Address	ASM	Bytes ^					^				
' (PopR 10 13503 810113)'	16r1000000	ld a0, 0(sp)	#[3 53 1 0]	lr			'16r1001000'		SP	16r1002FE8	16r1013400	
' (Label 1)'	16r1000004	addi sp, sp, 8	3 #[19 1 129 0]	рс			'16r1000'			16r1002FF0	16r1013400	
' (TstCqR 7 10 757D93)'	16r1000008	andi s11, a0,	#[147 125 11	sp			'16r1002FE8'			16r1002FF8	16r1013400	
' (JumpNonZero (Label 2) 20D9063)'	16r100000C	bnez s11, 32	#[99 144 13 2	fp			'16r1003000'		FP	16r1003000	16r0	
' (MoveMwrR 0 10 22/16 53B03)'	16r1000010	ld s6, 0(a0)	#[3 59 5 0]	x0	zero		'16r0'			16r1003008	16r0	
' (AndCqR 4194295/3FFFF7 22/16 no mcode)'	16r1000014	lui t0, 1024	#[183 2 64 0]	x1	ra		'16r1001000'			16r1003010	16r0	
' (JumpNonZero (Label 2) D9663)'	16r1000018	addiw t0, t0,	-#[155 130 11	x2	sp	sp	'16r1002FE8'			16r1003018	16r0	
' (MoveMwrR 8 10 10 853503)'	16r100001C	and s6, s6, t0	) #[51 123 91 (	х3	gp		'16r0'			16r1003020	16r0	
' (Jump (Label 1) FE1FF06F)'	16r1000020	bnez s11, 12	#[99 150 13 (	x4	tp		'16r0'			16r1003028	16r0	
' (Label 2)'	16r1000024	ld a0, 8(a0)	#[3 53 133 0]	x5	t0  i	ip1	'16r0'			16r1003030	16r0	
' (MoveMwrR 0 2 23/17 13B83)'	16r1000028	j -32	#[111 240 31	x6	t1  i	ip2	'16r0'			16r1003038	16r0	
' (Label 3)'	16r100002C	ld s7, 0(sp)	#[131 59 1 0]	х7	t2		'16r0'			16r1003040	16r0	
' (TstCqR 7 23/17 7BFD93)'	16r1000030	andi s11, s7,	1#[147 253 12	x8	s0(fp)	fp	'16r1003000'			16r1003048	16r0	
' (JumpNonZero (Label 4) 20D9063)'	16r1000034	bnez s11, 32	#[99 144 13 :	x9	s1		'16r0'			16r1003050	16r0	
' (MoveMwrR 0 23/17 22/16 BBB03)'	16r1000038	ld s6, 0(s7)	#[3 187 11 0]	x10	a0	arg0	'16r0'			16r1003058	16r0	
' (AndCqR 4194295/3FFFF7 22/16 no mcode)'	16r100003C	lui t0, 1024	#[183 2 64 0]	x11	a1	arg1	'16r0'			16r1003060	16r0	
' (JumpNonZero (Label 4) D9663)'	16r1000040	addiw t0, t0,	-#[155 130 11	x12	a2	carg0	'16r0'			16r1003068	16r0	
' (MoveMwrR 8 23/17 23/17 8BBB83)'	16r1000044	and s6, s6, t0	) #[51 123 91 (	x13	a3	carg1	'16r0'			16r1003070	16r0	
' (Jump (Label 3) FE1FF06F)'	16r1000048	bnez s11, 12	#[99 150 13 (	x14	a4	carg2	'16r0'			16r1003078	16r0	
' (Label 4)'	16r100004C	ld s7, 8(s7)	#[131 187 13	x15	a5	carg3	'16r0'			16r1003080	16r0	
' (CmpRR 10 23/17 41750DB3)'	16r1000050	j -32	#[111 240 31	x16	a6		'16r0'			16r1003088	16r0	
' (JumpNonZero (MoveCqR 16856080/1013410 10	16r1000054	sub s11, a0, s	s #[179 13 117	x19	s3	extral	'16r0'			16r1003090	16r0	
' (MoveCqR 16856096/1013420 10 1013537 42050!	16r1000058	bnez s11, 16	#[99 152 13 (	x20	s4	extra2	'16r0'			16r1003098	16r0	
' (Jump (MoveRMwr 10 0 2 A13023) C0006F)'	16r100005C	lui a0, 4115	#[55 53 1 1] 🗸	x22	s6	temp	'16r0'	~		16r10030A0	16r0	
Jump to		Step					Disassemble at PC					





## **Pharo VM Manual Variable Localisation**



### Automatic Variable localisation! Intel x86-64



Averages of 100 iterations + stdev. Relative to baseline (no optimisation). Higher



## **Analysing Code Cache Behavior**





### **Code Cache Unexpected results**



**Code Cache Size** 



Young Space ----- 1MB ---- 10 MB ---- 100 MB

**Loading Moose** 

## We are hiring!

- We have
  - Engineer Positions
  - Phd Positions

• Keywords: Compilers, Interpreters, Memory Management, Security

Come talk to us!



## Pharo VM - News from the Front



Université de Lille











Permanent Space Ephemerons

### New Image Format

**b** Lifeware



Speculative Compilation

### guillermo.polito@univ-lille.fr

@pharoproject

pharo.org

consortium-adm@pharo.org

- discord.gg/QewZMZa
  - thepharo.dev

53



## Conclusion

- 478 differences found, 91 causes, 6 categories
- Practical:
  - 4928 tests generated in ~8 minutes



Gaille Potito rupiable Tessones - Stéphane Ducasse guillermo.polito@univ-lille.fr @guillep 54



## Improvements: Clean Up

- V3 Support
- Old Memory Format
- Old Block Closures
- Dead Code
- ~ 65KLOC

## **Improvements: Sockets**

- Unified Implementation in all Platforms
- Better Async Support
- Unix Sockets (Under Work)
- IPv6 Addresses (Under Work)

## **Improvements: Serial Port FFI**

- Pure FFI implementation
- Working in all Platforms (Unix / Windows / OSX)
- Migrating Plugins to FFI

### Improvements: RISCV64 Ongoing Port

- Currently under development: Real HW testing stage
- Taking advantage of our harness test suite.
- Improving tests and scenarios
- Collaboration with Q. Ducasse, P. Cortret, L. Lagadec from ENSTA Bretagne
- Future work on: Hardware-based security enforcement



### Improvements: Open Build Service Better Support for Linux Distributions

		Arch	ì	Debian_	10	Debian_9.0	Debian_Testing	Fedora_	31	Fedora_32	Fedora_	33	Raspl	pian_10	Raspbia	n_9.0
	↑↓	🛼 x86	_64∿	🛼 x86_	<b>64</b> U	, x86_64∿	🔜 x86_64 🎊	🛼 x86_	64∿	😝 x86_64∜	🛼 x86_	.64∿	🛼 aarch64	k 💀 x86_64∜	, aarch64 🕁	, x86_64⊍
libffi7				succee	ded	succeeded		succee	ded	succeeded	succeede		succeeded	succeeded	succeeded	succeeded
libgit2-1				succee	ded		failed									
pharo9		failed		succeeded failed		failed	failed	failed failed		failed	failed		succeeded	succeeded	failed	failed
pharo9-ui		succeeded succeeded succ		succeeded	failed succeeded		ded	succeeded	succeeded			succeeded		succeeded		
			k													
		Raspbia	n_9.0		opens	SUSE_Leap_15	5.1 openSUSE_I	.eap_15.2	ope	enSUSE_Tumble	eweed	xUbu	untu_18.04	xUbuntu_19.04	xUbunt	u_20.04
	↑↓	arch64 🏼 🛼		x86_64\/ 🛼 x86_64		⇒ x86_64	11 💀 x86	6 <b>4</b> ↑↓		🛼 x86_64	↑↓		x86_64 ↑↓	🛼 x86_64 🛝	🖶 aarch64	🛼 x86_64
ibffi7	:	ceeded	suc	ceeded		succeeded	succe	eded		succeeded		succeeded		succeeded	succeeded	succeeded
ibgit2-1						succeeded	succe	eded				succeeded		succeeded		succeeded
oharo9		failed	fa	ailed		failed	fail	ed		failed			failed	succeeded	succeeded	succeede
pharo9-ui			suc	ceeded		succeeded	succe	eded		succeeded		S	ucceeded	succeeded		succeede

Initial targets:

- Arch / Manjaro
- Debian
- Fedora
- Raspbian
- Ubuntu
- openSuse

Building using existing system

Supporting system

Multiple Architectures

### Improvements: Visual Studio Support Building & Debugging

• • • • • • • • • • • • • • • • • • •	▶ Eile Edit View Git Project Build	Debug         Test         Analyze         Jools         Extensions         Window         Help         Search (Ctrl+Q)         P         PharoVM		G – 🗆 >
Prese [b229] Processes 1 (bdopske bask / 1 house) (bdopske bask / 1 hou	🖗 🗢 🗢 🛛 🔁 🖕 🔛 🚰 😕 - 🕫 - 🛛 De	bug 🔹 🗚 M64 🔹 🕨 Continue * 🏓 🙆 🔹 🔹 🖉		$\blacksquare \bullet \bullet \Rightarrow \ddagger ? \ddagger \ddagger \Downarrow B \text{ Live Share } \mathscr{F}$
bedeta bedeta	Process: [0x2730] PharoConsole.exe *	Lifecycle Events * Thread: 🚽 🐺 🕫 223 Stack Frame:	~ <del>.</del>	
sign:	Registers         X0 = 0000000135715120 X1 = 00000000000           X6 = 00000167520AA28 X7 = 00000000000         X12 = 000000075081000 X13 = 00000000           X18 = 00000037C9E81000 X13 = 00000000         X24 = 00000000000000 X25 = 00000000           X24 = 000000000000000 X25 = 00000000         LR = 000007C9E81000 X13 = 00000007           X05 = 0000070FF987993CE8 SP = 00000007         FI           Disassembly         sqWin32Directory.c         FI           SFIEleAttributesPlugin         578         primitiveFileAttribute(void)           578         [         fileAttribute(void)	8109 X2 = 00007FF9755F8830 X3 = 00000000000067 X4 = 00007FF9755F8814 X5 = 00007FF9755F8798           900000 X8 = 000000000000000 X9 = 00000000000008 X10 = 00007FF9756218 X11 = 000000000040           9000001 X14 = 00000000000000 X1 = 000007F0755F8814 X5 = 000007FF97568218 X11 = 0000000000040           9000001 X14 = 00000000000000 X21 = 0000000000001832 B1 P1 = 000000000000000000000000000000000	Prop Pha Pha Bit N Pr Pr Pr Pr Pr Pr Pr Pr Pr Pr Pr Pr Pr	ertie Solution Explorer
Image: Notice Notice Notice       Image: Notice Notice Notice       Image: Notice Noti	S81     fapath faPath;       C:\Users\\se\Download>\tfn\tfn\tfn\tfn\tfn\tfn\tfn\tfn\tfn\tfn	ing Windows Help	-	D faCommon.c     D faSupport.c     D fieldtributesPlugin.c     GriedtributesPlugin     MakeLists.tot     Torrestype     Torrestype     Torrestype     Torrestype
Playground UndefinedObject      IndefinedObject      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Add to Source Control      Add to Source Control      Add to Source Control      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Add to Source Control      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Add to Source Control      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Add to Source Control      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Add to Source Control      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W Output Error List      Call Stack Breakpoints Exception Set Command W Immediate W O	100 Puttin Bindig: Page	x - □       UndefinedObject       •         Models       V © ProtoObject       • instance side       •         Numbers       V © PotoObject       • instance side       •         Objects       V © Dolean       • Elextensions       addSubclass:         Pragmas       © False       © True       bottom context       class hierarchy       • addDependent:         Processes       © True       Elter       • Dottom context       class hierarchy       • addDependents         • AllPackages O Scoped View   ● Flat O Hier.   ● Inst. side O Class side   ● Methods O Vars   Class       • MIXED         ? Comment × © UndefinedObject       A UML-Class × + Inst. side methc × +       +         Object subclass: #UndefinedObject       · LUML-Class × + Inst. side methc × +       +         object subclass: #UndefinedObject       · classVariableNames: ''       · classVariableNames: ''	CRLF # ×	<ul> <li>▷ Sol gitZinternal</li> <li>▷ Sol http:-parser</li> <li>▷ Sol http:-parser</li> <li>▷ Sol JPEGReadWriter2Plugin</li> <li>▷ Sol JPEGReadWriter2Plugin</li> <li>▷ Sol JEGReadWriter2Plugin</li> <li>▷ Sol LocalePlugin</li> <li>▷ Sol LocalePlugin</li> <li>▷ Sol MiscPrimitivePlugin</li> <li>▷ Sol ACKAGE</li> <li>▷ Sol Pharo</li> <li>▷ Sol Pharo</li> <li>▷ Sol Pharo</li> <li>▷ Sol Solz</li> <li>▷ Solz</li> <l< td=""></l<></ul>
Add to Source Control +      Add to Source Control +      1225	Playground UndefinedObject	Call Stark, Breakpoints, Evrention Set, Command W, Immediate W, Output, Er	v (Na Spec	b         Gg. SurfacePlugin           b         Gg. FestLibrary           b         Gg. UUIDPlugin           b         Gg. ZERO_CHECK           offers         b           cifes         b           b         Gg. ZERO_CHECK
	□ Ready	сполосите структи сокрытисти та тапесное на сокрыте		🕈 Add to Source Control 🔺 🦾
	C Sourch			

MSVC - No cygwin

### **Improvements: Windows ARM**

	gging Windows Help							_			
	Disustant			Test R	unner			r			
	Playground			×	274 ran, 270 passe	ed, 0 skipped, 3 expect	ted failures, 0	^			
oit Publish Bindings Pages				^	failures, 1 error, 0			~			
1 canvas := RSCanvas new.		× - 🗆	- CmdMenu>>asSpMenuPresenter								
2 shapes := #(20 10 5 30 24 32	) collect: [ :e   RSEllipse model:	e ] BaselineOfSpec2	^ (C	CmdComn	nand	instance side	· ·	asSpMenuPresenter			
3 canvas addAll: shapes.	anacı parmalizat tuqurcalf	Commander-Spec2-	Compatit ©	CmdComm	nandActivator	extensions					
5 RSElowLayout new alignCenter	ont shapes.	Spec2-Adapters-More	phic C	CmdComm	nandMenuItem	🗈 Commander	-Spec2-Compatil				
6 canvas @ RSCanvasController	, on anapea.	Spec2-Adapters-More	phic-Test 📀	CmdMenu		accessing					
		Spec2-Adapters-Stu	b C	CmdMenu	Group	building					
		Spec2-Backend-Test	5			<ul> <li>initialization</li> </ul>					
		▶ Spec2-Code				testing					
		specz	✓ Filter.			overrides	~				
- 🗆	Inspector on a RSCanvas	All Packages O Scope	d View   O Flat	O Hier.	Inst. side O Cl	ass side   • Method	Is O Vars   <u>Class r</u>	efs. Q Implementors	(), <u>Sende</u>		
RSCanvas		? Comment ×	CmdMenu	×	asSpMenuPrese ×	UML-Class	× + Inst. side	methc ×	+ +		
anvas Raw Shapes Items Meta									He		
					C	mdMenu					
					activation Strategy	Class					
					activation Strategy	Class					
					activationStrategy	Class:					
					asSpMenuPresente	Hr.					
					buildBrowserTabA	ctions					
					buildBrowserToolb	xar:					
					buildContextMenu	mincontext activatingву: For:					
			× - 🗆			System Repo	rter				
			Image	^ (	Operating System/H	ardware					
			Image Paramete	rs -							
			MC Repositories	,	Win32 Windows-ARM	164 ARM64					
			MC Working Con	ios							
			VM General								
			VM Options								
			VM Modules								
			VM Parameters								
			VM Stats								
			OS								
			OS Environment								
			OS Details	*							
									_		
Test Runner	Prese E Playground 6 Inspector on a RSCa	nvas System Reporter									
Test Runner 🔐 CmdMenu>>asSpMenu	Prese 🔄 Playground 🎸 Inspector on a RSCa	invas 🔄 System Reporter		•		_	ê 🛎 📾 🖉	ξ q× g <sup>β</sup> ENG	15:08		

MSVC - No cygwin

### Improvements: Raspbian 32/64 bits

	••				192	.168.0	35:59	01 (raspber	rypi:1 (pi	)) - VNC	Viewer						
\$	Pharo	Tools	s System	Debug	ging Window	s H	elp										
	Pharo	Tools System Tools System Tage Image Image Parameters Image Sources MC Repositories MC Repositories MC Working Copies VM General VM Options VM Modules VM Parameters VM Stats		Operatin COLORTE DBUS_SE /session- DESKTOF DISPLAYE DISPLAYE GPG_AGE HOME=/H	ging Window System Reporte g System Environi ERM=truecolor ESSION_BUS_ADD ef2f7aa707634b96 P_SESSION=LXDE- ::1.0 NUM=1 ENT_INFO=/run/us nome/pi ME=receptorprini	s H r nent RE × · //t Doi Doi 1	- D F Pub 1 ti	Bindings nyBenchma	v ⇒≣ Pages Parks "'	7277896	Playg 23 byte	round codes/se	ec; 5	51028940	) send	ls/sec1"	•
		OS OS Environ OS Details	nment	HOSTNAI LANG=en LC_CTYP LC_TERM LD_LIBR/ b:/lib/aai r/lib: LOGNAM LS_COLO 35:do=01 =37;41:sg	ME=raspberrypi _GB.UTF-8 E=UTF-8 IINAL=iTerm2 IINAL_VERSION=3 ARY_PATH=/usr/lil rch64-linux-gnu:/li E=pi DRS=rs=0:di=01;34 ;35:bd=40;33;01:c g=30;43:ca=30;41:t	3. b: d= W											
				1;32:*.tar	=01;31:*.tgz=01;3	Line	:1:1										+L
				1;31:*.lha 01;31:*.b ;31:*.dz= *.xz=01;3 .tbz=01;3 :*.jar=01;	<pre>&gt;=01;31:*.lz4=01;3 cz=01;31:*.tzo=01; 01;31:*.gz=01;31:* 1:*.zst=01;31:*.tzs 1:*.tbz2=01;31:*.tz 31:*.war=01;31:*.e &gt;</pre>	l:1 31:*.t7z= lrz=01;3 t=01;31: z=01;31: sar=01;3 oo=01:7	=01;31:* )1:*.lz=( *.bz2=( *.deb=( 1:*.sar=	'.zip=01;31:* 01;31:*.lzo=0 01;31:*.bz=0) 01;31:*.rpm= =01;31:*.rar=	.z=01 11;31: 1;31:* 01;31 01;31 -01:3	Kernel er All Packa Comme nextI "Si (Smal (Smal	I-Tests-Extr nges O Sc ent × <b>nstance</b> mallInte lInteger mallInte	ended oped View © Sm egers ca r minVal eger min	r   ● mallinte an't L) to nVal)	Filter Flat O F eger x be enum (Small to: (S	Hier.   Y⊡⊓ erate Integ mallI	Inst.side nextInstance d this wai er maxVal nteger mai	O Class × 2 y. The ), but xVal) c
J	Reposi	tories 🧃	Working cop	of pharo	SmallInteger	>nextIr	ist [	System R	eporter	Playg	round						



## Back to the Future Objectives for 2022



### Permanent Space Problem



Generates

Long Pauses

- Many permanent objects
- They have references from/to other objects
- We are traversing them to GC
- E.g., Classes, Methods, Literals, Resources



### **Permanent Space** Our Solution



- New Object Space for permanent Objects
- Minimise or Eliminate GC passes
- Persisting them through executions



### **Permanent Space** Our Solution



Lifeware

- Minimise or Eliminate GC passes
- Persisting them through executions
   We need to put them in a

## New Image Format 🛛 🔈 Lifeware

### Problem

- Current Image format only support a single object space
- No extensible: not new metadata nor new data
- Cannot be Memory Mapped (it is modified before save/load)
- Requires to discard all state of the running VM (slow saves)

## New Image Format Lifeware

### Problem

- Current Image format only support a single object space
- No extensible: not new metadata nor new data
- Cannot be Memory Mapped (it is modified before save/load)
- Requires to discard all state of the running VM (slow saves)
   Slow and Restricting

### New Image Format Solution

- New Image format based in directories / bundles
- Many Elements of data and metadata
- Metadata en User & Machine readable format (STON?)
- Extensible format



### **Fast Snapshots / Loading** Based on PermSpace & Image Format

- Memory Mapped Image
- Shareable State
- Saving / Loading Warm State of the VM



### **Next Objectives**

Permanent Space Ephemerons New Image Format Speculative Faster Startup / Saving<sup>Compilation</sup>

- ARM64, RISCV64, Slang...
- Lots of Tests!
- @pharoproject
   pharo.org Integration: Sockets, serial
- Visual Studio, OperoBsiddiServicem@pharo.org discord.gg/QewZMZa thepharo.dev

# **Cross-ISA Testing of the Pharo VM**

### Lessons learned while porting to ARMv8 64bits Tool Paper — MPLR'21

Guille Polito, Stéphane Ducasse, Pablo Tesone, Théo Rogliano, Pierre Misse-Chanabier, Carolina Hernandez, Luc Fabresse RMoD Team – Inria Lille Nord Europe – UMR9189 CRIStAL – CNRS














## **Some Numbers**

- 255 stack based bytecodes (77 different) + ~340 primitives/ native methods
   Lots of combinations!
- 146 different IR instructions
- polymorphic inline caches
- threaded code interpreter
- generational scavenger GC

#### **Objective: Implementing an ARM64 Backend**

- ARM64 is now pervasive move r1 #1
  - New Apple M1
  - Raspberry Pi 4
  - Microsoft Surface Pro
  - PineBook Pro

move r1 #1 move r2 #17 checkSmallInt checkSmallInt add r3 r1 r2 checkSmallInt move r1 r3 ret

JIT compiler IR



#### Case Study 1 Porting the Cogit JIT Compiler to ARM64

- Started with no tests and no hardware (main target Apple M1)
- Incremental test development: bytecode, native methods, PICs, code patching
- All tests run from the beginning on our four targets: x86, x86-64, ARM32 and ARM64
- Test allowed safe modifications in the IR to support e.g., ARM64 Multiplication overflow
- ARM64 specific tests covered stack alignment, W+X ...



#### Case Study 2 Ongoing Port to RISCV64

- Currently under development
- Is our harness test suite enough to develop a new backend?
- Are our tests general enough?
- Collaboration with Q. Ducasse, P. Cortret, L. Lagadec from ENSTA Bretagne
- Future work on: Hardware-based security enforcement



#### Case Study 3

#### **Debugging and Testing Memory Corruptions**

- Bug report using Ephemerons https://github.com/pharo-project/pharo/issues/8153
- Starting the other way around
  - First reproducing the bug in real-hardware
     => long to execute (even longer in simulation)
     => required manual developer intervention
  - Then building a unit test from observations
  - Test becomes a part of the regression test suite

#### **Future Perspectives**

Automatic VM Validation

- Automatic (Unit?) Test Case Generation
- Interpreter vs Compiler Differential Testing
- VM Tailored Multi-level Debugging

# Cross-ISA Testing of the Phase Med while porting to ARMv8 64bits

	Real Hardware Execution	Full-System Simulation	Unit-Testing
Feedback-cycle speed	Very low	Low	High
Availability	Low	High	High
Reproducibility	Low	Low	High
Precision	High	Low	Low
Debuggability	Low	High	High





#### **Debugging a compiler** Insights: build your own tools, based on needs, not desires

× - 🗆			VM De	bugger						
IR Instructions	Address	ASM	Bytes ^					^		
(PopR 10 13503 810113)'	16r1000000	ld a0, 0(sp)	#[3 53 1 0]	lr			'16r1001000'	SP	16r1002FE8	16r101340
' (Label 1)'	16r1000004	addi sp, sp, 8	3 #[19 1 129 0]	рс			'16r1000'		16r1002FF0	16r101340
' (TstCqR 7 10 757D93)'	16r1000008	andi s11, a0,	#[147 125 11	sp			'16r1002FE8'		16r1002FF8	16r101340
' (JumpNonZero (Label 2) 20D9063)'	16r100000C	bnez s11, 32	#[99 144 13 :	fp			'16r1003000'	FP	16r1003000	16r0
' (MoveMwrR 0 10 22/16 53B03)'	16r1000010	ld s6, 0(a0)	#[3 59 5 0]	x0	zero	)	'16r0'		16r1003008	16r0
' (AndCqR 4194295/3FFFF7 22/16 no mcode)'	16r1000014	lui t0, 1024	#[183 2 64 0]	x1	ra		'16r1001000'		16r1003010	16r0
' (JumpNonZero (Label 2) D9663)'	16r1000018	addiw t0, t0,	-#[155 130 11	x2	sp	sp	'16r1002FE8'		16r1003018	16r0
' (MoveMwrR 8 10 10 853503)'	16r100001C	and s6, s6, t0	) #[51 123 91 (	х3	gp		'16r0'		16r1003020	16r0
(Jump (Label 1) FE1FF06F)'	16r1000020	bnez s11, 12	#[99 150 13 (	x4	tp		'16r0'		16r1003028	16r0
(Label 2)'	16r1000024	ld a0, 8(a0)	#[3 53 133 0]	x5	t0	ip1	'16r0'		16r1003030	16r0
' (MoveMwrR 0 2 23/17 13B83)'	16r1000028	j-32	#[111 240 31	x6	t1	ip2	'16r0'		16r1003038	16r0
(Label 3)'	16r100002C	ld s7, 0(sp)	#[131 59 1 0]	x7	t2		'16r0'		16r1003040	16r0
(TstCqR 7 23/17 7BFD93)'	16r1000030	andi s11, s7,	1#[147 253 12	x8	s0(f	p)   fp	'16r1003000'		16r1003048	16r0
(JumpNonZero (Label 4) 20D9063)'	16r1000034	bnez s11, 32	#[99 144 13 :	x9	s1		'16r0'		16r1003050	16r0
(MoveMwrR 0 23/17 22/16 BBB03)'	16r1000038	ld s6, 0(s7)	#[3 187 11 0]	x10	a0	arg0	'16r0'		16r1003058	16r0
' (AndCqR 4194295/3FFFF7 22/16 no mcode)'	16r100003C	lui t0, 1024	#[183 2 64 0]	x11	al	arg1	'16r0'		16r1003060	16r0
(JumpNonZero (Label 4) D9663)'	16r1000040	addiw t0, t0,	·#[155 130 11	x12	a2	carg0	'16r0'		16r1003068	16r0
(MoveMwrR 8 23/17 23/17 8BBB83)'	16r1000044	and s6, s6, t0	) #[51 123 91 (	x13	a3	carg1	'16r0'		16r1003070	16r0
' (Jump (Label 3) FE1FF06F)'	16r1000048	bnez s11, 12	#[99 150 13 (	x14	a4	carg2	'16r0'		16r1003078	16r0
(Label 4)'	16r100004C	ld s7, 8(s7)	#[131 187 13	x15	a5	carg3	'16r0'		16r1003080	16r0
(CmpRR 10 23/17 41750DB3)'	16r1000050	j-32	#[111 240 31	x16	a6		'16r0'		16r1003088	16r0
(JumpNonZero (MoveCqR 16856080/1013410 10	16r1000054	sub s11, a0, s	s #[179 13 117	x19	s3	extra1	'16r0'		16r1003090	16r0
(MoveCqR 16856096/1013420 10 1013537 42050	16r1000058	bnez s11, 16	#[99 152 13 (	x20	s4	extra2	'16r0'		16r1003098	16r0
' (Jump (MoveRMwr 10 0 2 A13023) C0006F)'	16r100005C	lui a0, 4115	#[55 53 1 1] 🗸	x22	s6	temp	'16r0'	~	16r10030A0	16r0
Jump to		81	Step						Disassemble at PC	

#### Examples:

- Machine code debugger
- Bytecode-IR visualization
  Disassembler
  DSL

## Interpreter-Guided JIT Compiler Test Generation

# Validating the Pharo JIT compiler through concolic execution and differential testing

Guille Polito - Pablo Tesone - Stéphane Ducasse guillermo.polito@univ-lille.fr @guillep





#### PLDI'22 – San Diego















#### How can we automatically test VMs?









#### Interpreter-Guided Automatic JIT Compiler Unit Testing













## **Concolic Meta-Interpretation Model**

- Models VM behaviour during concolic execution
  - Frame
  - Objects + types
  - Classes
- Then flattened into SAT solver equations



## **Experimental Context: The Pharo VM**

- Interpreted-compiled mixed execution
- Some numbers:
  - 255 stack based bytecodes
  - ~340 primitives/native methods
  - 146 different IR instructions
  - x86, x86-64, ARMv7, ARMv8, RISC-V
- Industrial consortium:
  - 28 International companies, 26 academic partners



#### **Previous Manual Testing Effort**

- No useful unit tests by ~06/2020
- Large manual testing effort during 2020 while porting to ARM64bits
  - Extended VM simulation with a (TDD compatible) unit testing infrastructure
  - 450+ written tests on the interpreter and the garbage collector\*
  - 580+ written tests on the JIT compiler\*
  - Parametrisable for 32 and 64bits, ARM32, ARM64, x86, x86, d4bers by 05/20
     Cross-ISA Testing of the Phago VM. Lessons learned while porting to ARMv8 64bits.



#### **Evaluation**

- 3 bytecode compilers + 1 native method compiler
- 4928 tests generated

170

Compiler	# Tested Instructions	# Interpreter Paths	# Curated Paths	# Differences (%)
Native Methods (primitives)	112	2024	1520	440 (28,95%)
Simple Stack BC Compiler	175	1308	1136	18 (1,59%)
Stack-to-Register BC Compiler	175	1308	1136	10 (0,88%)
Linear-Scan Allocator BC Compiler	175	1308	1136	10 (0,88%)
Total	637	5948	4928	478 (9,7%)



#### **Analysis of Differences through Manual Inspection**

- 91 causes, 6 different categories
- Errors both in the interpreter AND the compilers

•	1/ causes of <b>segmentation</b>	Family	# Cases
-	14 Causes of Segmentation	Missing interpreter type check	1
		Missing compiled type check	13
		Optimisation difference	10
		Behavioral difference	5
		Missing Functionality	60
		Simulation Error	2
1			

## **Characterising Concolic Execution**

103

Paths per instruction

- Native methods present in average more paths than bytecode instructions
  - => longer time to explore
  - => potentially more bugs



#### **Practical and Cheap**

- Test generation ~5 minutes
- Total run time of ~10 seconds
  Avg 30ms per instruction





#### More in the article!

- **Discovered Bugs**
- Concolic Model
- Testing Infrastruct

#### **Interpreter-Guided Differential JIT Compiler Unit** Testing

Guillermo Polito Univ. Lille, CNRS, Inria, Centrale Lille, Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille France guillermo.polito@univ-lille.fr

Stéphane Ducasse UMR 9189 CRIStAL France stephane.ducasse@inria.fr

Pablo Tesone Pharo Consortium Univ. Lille, Inria, CNRS, Centrale Lill UMR 9189 CRIStAL France pablo.tesone@inria.fr

#### Abstract

Modern language implementations using Virtual Machines feature diverse execution engines such as byte-code interpreters and machine-code dynamic translators, a.k.a. JIT compilers. Validating such engines requires not only validating each in isolation, but also that they are functionally equivalent. Tests should be duplicated for each execution engine, exercising the same execution paths on each of them.

In this paper, we present a novel automated testing approach for virtual machines featuring byte-code interpreters. Our solution uses concolic meta-interpretation: it applies concolic testing to a byte-code interpreter to explore all possible execution interpreter paths and obtain a list of concrete values that explore such paths. We then use such values

San Diego, CA, USA. ACM, New York, NY, USA, 12 pages. //doi.org/10.1145/3519939.3523457

#### **1** Introduction

Modern Virtual Machines support code generation fo compilation and dynamic code patching for techniques as inline caching. They are often structured around a code interpreter, a baseline JIT compiler, and a specula inliner. This complexity is aggravated when the VM by and runs on multiple target architectures [1]. Validating execution of interpreted code and its compiled counter is challenging.

Several solutions have been proposed to aid in VM tes tasks. Traditionally, VM simulation environments hav



## Conclusion

- 478 differences found, 91 causes, 6 categories
- Practical:
  - 4928 tests generated in ~8 minutes



Gaille Polito rup able Tossones - Stéphane Ducasse guillermo.polito@univ-lille.fr @guillep

# Extras



## Simulation + Testing Environment

	VM			
		GC	JIT Compiler	Transpiled to
infrastructure	Неар			
	Native Code Cache			
				Production VN
Unicorn	LLVM			


## **Unit Testing Infrastructure Comparison**

	Real Hardware Execution	Full-System Simulation	Unit-Testing
Feedback- cycle speed	Very low	Low	High
Availability	Low	High	High
Reproducibilit y	Low	Low	High
Precision	High	Low	Low
Debuggability	Low	High	High



