



# Bugloo: A Source Level Debugger for Scheme Programs Compiled into JVM Bytecode

Damien Ciabrini

Manuel Serrano

`firstname.lastname@sophia.inria.fr`

INRIA Sophia Antipolis

2004 route des Lucioles - BP 93

F-06902 Sophia Antipolis, Cedex



# Outline

---



- Introduction
- The Bugloo debugger
- Custom debugging features
- JVM debugging architecture
- Mapping Scheme to JVM
- Performances
- Conclusion



# Introduction

---



- Debugging programs:
  - to detect, to locate and to corrects errors
- Two kinds of debuggers:
  - static debuggers
  - dynamic debuggers



# Motivation of our Work



- Programmers hardly use debuggers:
  - sometimes not efficient enough
  - not adapted to correct certain bugs
  - "prints are simpler and quicker"
- How to make debuggers more attractive ?



# Motivation of our Work



- Programmers hardly use debuggers:
  - sometimes not efficient enough
  - not adapted to correct certain bugs
  - "prints are simpler and quicker"
- How to make debuggers more attractive ?
  - easily accessible from the IDE
  - acceptable performance slowdown
  - to deal with the language specificities



# Context of Development



- Work with the **Bigloo** Scheme compiler:
  - Scheme  $\Rightarrow$  C. Already has a debugger.
  - Scheme  $\Rightarrow$  JVM bytecode. JVM provides JPDA.
- JVM Platform Debugging Architecture (JPDA):
  - A set of APIs to make debugger and profilers
  - Standardized  $\Rightarrow$  portability across JVMs
  - JIT can limit performances slowdown
  - Same classfile for normal or debug executions



# The Bugloo Debugger

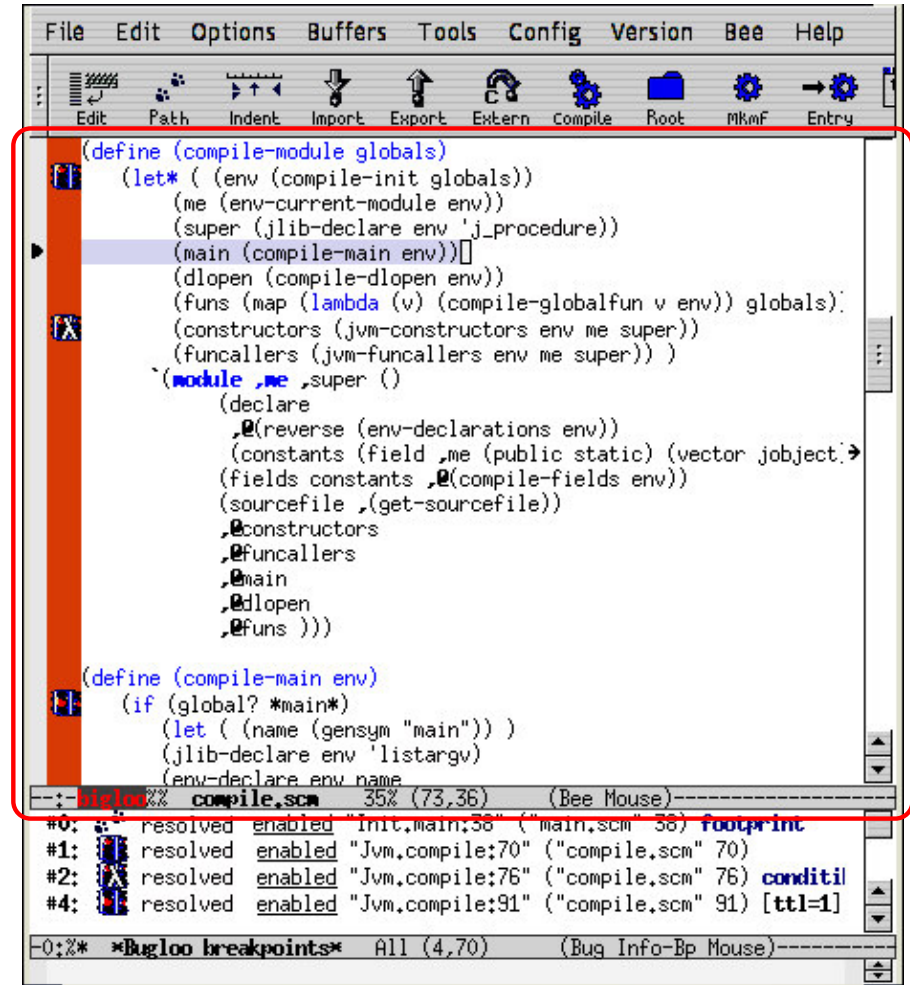


- Bugloo is a *source level debugger* :
  - Operates on compiled Bigloo programs
- Basic instrumentation of the *debuggee* :
  - breakpoints, stack and variables inspection
- Advanced debugging features
  - Traces, memo-conditions, memory debugging
- Controlled by a command language
- Integrated into the Emacs editor



# Emacs Environment (1/2)

- Source buffers:



```
(define (compile-module globals)
  (let* ( (env (compile-init globals))
         (me (env-current-module env))
         (super (jlib-declare env 'j_procedure))
         (main (compile-main env))
         (dlopen (compile-dlopen env))
         (funs (map (lambda (v) (compile-globalfun v env)) globals))
         (constructors (jvm-constructors env me super))
         (funcallers (jvm-funcallers env me super)) )
    `(module ,me ,super ()
      (declare
        ,(reverse (env-declarations env))
        (constants (field ,me (public static) (vector jobject)))
        (fields constants ,(compile-fields env))
        (sourcefile ,(get-sourcefile))
        ,(compile-constructors)
        ,(compile-funcallers)
        ,(compile-main)
        ,(compile-dlopen)
        ,(compile-funs )))

  (define (compile-main env)
    (if (global? *main*)
        (let ( (name (gensym "main")) )
            (jlib-declare env 'listargv)
            (env-declare env name
```

--: Bugloo compile.scm 35% (73,36) (Bee Mouse)-----

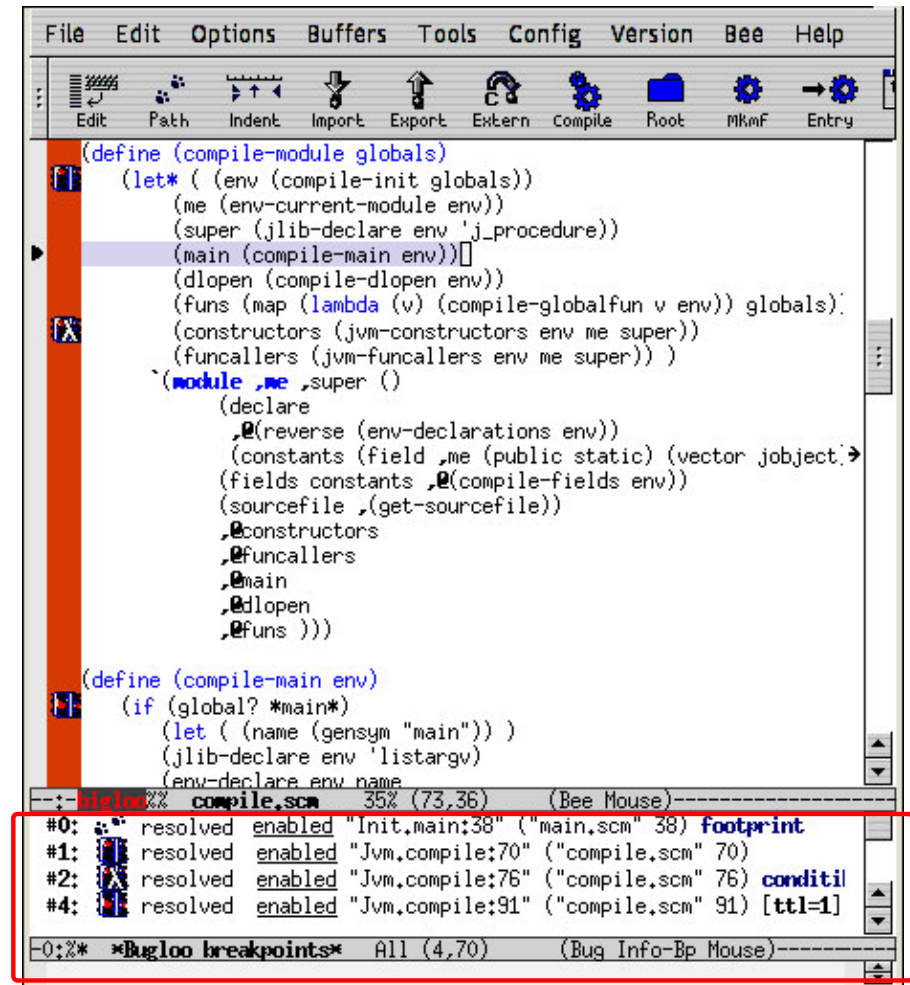
#0:	resolved	enabled	"Init.main:58" ("main.scm" 58)	footprint
#1:	resolved	enabled	"Jvm.compile:70" ("compile.scm" 70)	
#2:	resolved	enabled	"Jvm.compile:76" ("compile.scm" 76)	conditil
#4:	resolved	enabled	"Jvm.compile:91" ("compile.scm" 91)	[ttl=1]

-0:~\* Bugloo breakpoints\* All (4,70) (Bug Info-Bp Mouse)-----



# Emacs Environment (1/2)

- Source buffers:
  - Breakpoints list



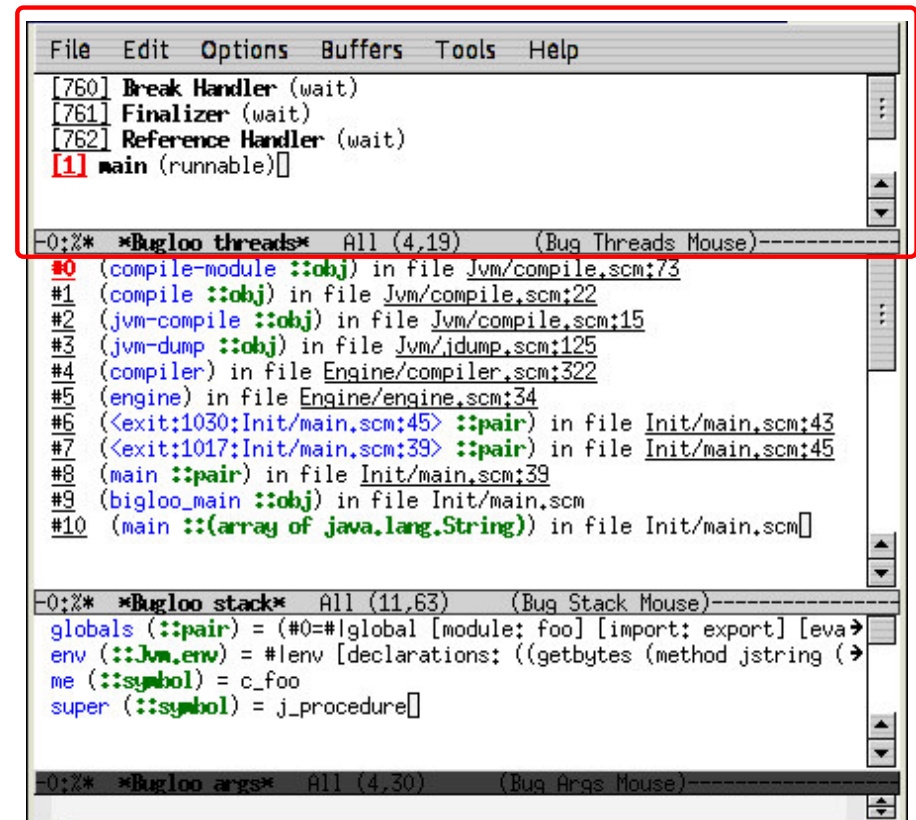
The screenshot shows the Emacs editor interface with a menu bar (File, Edit, Options, Buffers, Tools, Config, Version, Bee, Help) and a toolbar. The main window displays Scheme code for a module named 'compile'. The code includes a `(define (compile-module globals))` function that sets up an environment with various procedures and constants. Below the code, the Bugloo breakpoints list is visible, showing four breakpoints:

Breakpoint ID	Source	Location	File	Line	Condition	Other
#0	resolved	enabled	"Init.main:38"	("main.scm" 38)	footprint	
#1	resolved	enabled	"Jvm.compile:70"	("compile.scm" 70)		
#2	resolved	enabled	"Jvm.compile:76"	("compile.scm" 76)	conditil	
#4	resolved	enabled	"Jvm.compile:91"	("compile.scm" 91)	[ttl=1]	

At the bottom, there is a summary line: `*Bugloo breakpoints* All (4,70) (Bug Info-Bp Mouse)`. A red box highlights the breakpoints list.

# Emacs Environment (1/2)

- Source buffers:
  - Breakpoints list
- Execution State:
  - Debuggee inspection



The screenshot displays the Emacs Bugloo debugger interface. At the top, a menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', and 'Help'. Below the menu, a list of threads is shown, with the 'main' thread selected and highlighted in red. The thread list includes:

- [760] Break Handler (wait)
- [761] Finalizer (wait)
- [762] Reference Handler (wait)
- [1] main (runnable)

Below the thread list, the 'Bugloo threads' window is open, showing a list of threads with their IDs and descriptions:

- #0 (compile-module ::obj) in file Jvm/compile.scm:73
- #1 (compile ::obj) in file Jvm/compile.scm:22
- #2 (jvm-compile ::obj) in file Jvm/compile.scm:15
- #3 (jvm-dump ::obj) in file Jvm/jdump.scm:125
- #4 (compiler) in file Engine/compiler.scm:322
- #5 (engine) in file Engine/engine.scm:34
- #6 (<exit:1030:Init/main.scm:45> ::pair) in file Init/main.scm:43
- #7 (<exit:1017:Init/main.scm:39> ::pair) in file Init/main.scm:45
- #8 (main ::pair) in file Init/main.scm:39
- #9 (bigloo\_main ::obj) in file Init/main.scm
- #10 (main ::(array of java.lang.String)) in file Init/main.scm

Below the threads window, the 'Bugloo stack' window is open, showing the current stack frame:

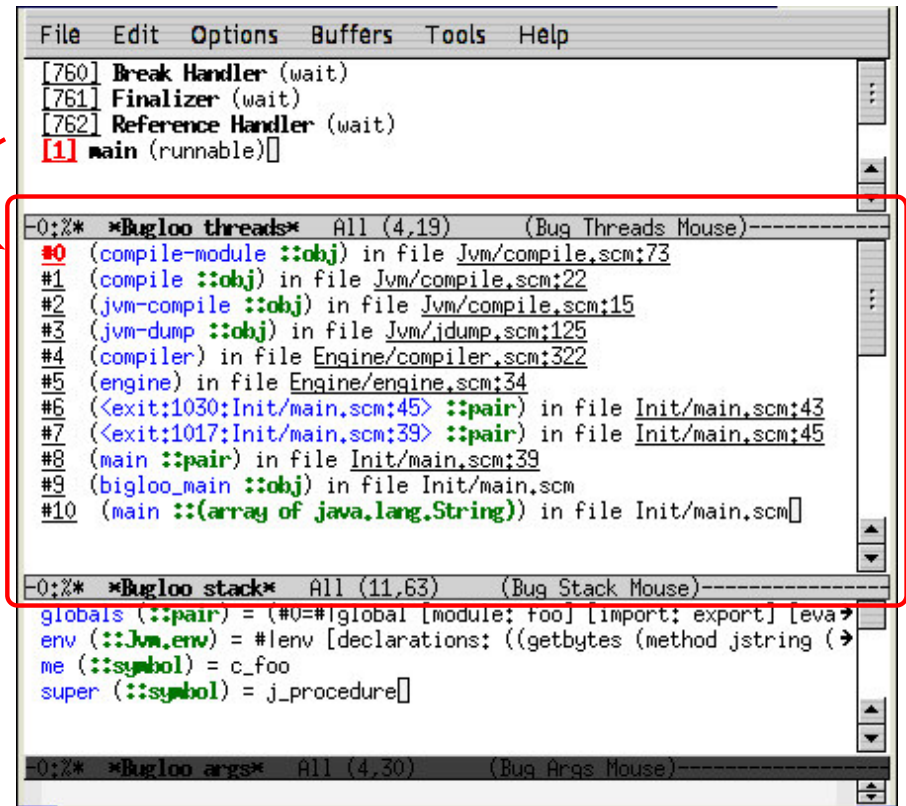
- globals (::pair) = (#0=#global [module: foo] [import: export] [eva
- env (::Jvm.env) = #lenv [declarations: ((getbytes (method jstring (
- me (::symbol) = c\_foo
- super (::symbol) = j\_procedure

At the bottom, the 'Bugloo args' window is open, showing the current arguments:

- All (4,30) (Bug Args Mouse)

# Emacs Environment (1/2)

- Source buffers:
  - Breakpoints list
- Execution State:
  - Debuggee inspection



The screenshot shows the Emacs Bugloo interface. At the top, there is a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', and 'Help'. Below the menu bar, a list of threads is displayed: [760] Break Handler (wait), [761] Finalizer (wait), [762] Reference Handler (wait), and [1] main (runnable). A red arrow points from the 'main' thread to the 'Bugloo threads' window. The 'Bugloo threads' window shows a list of threads from #0 to #10, including compile-module, compile, jvm-compile, jvm-dump, compiler, engine, and main. Below this, the 'Bugloo stack' window shows the stack for the main thread, including globals, env, me, and super. At the bottom, the 'Bugloo args' window shows the arguments for the main thread.

```
File Edit Options Buffers Tools Help
[760] Break Handler (wait)
[761] Finalizer (wait)
[762] Reference Handler (wait)
[1] main (runnable)

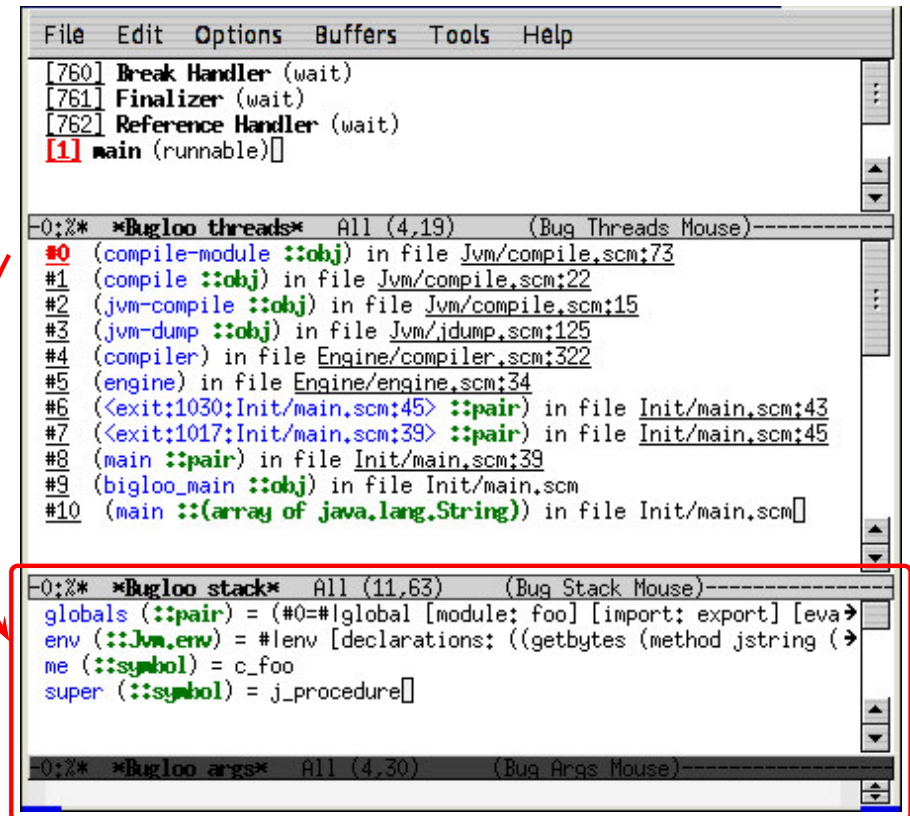
-0:2* *Bugloo threads* All (4,19) (Bug Threads Mouse)-----
#0 (compile-module ::obj) in file Jvm/compile.scm:73
#1 (compile ::obj) in file Jvm/compile.scm:22
#2 (jvm-compile ::obj) in file Jvm/compile.scm:15
#3 (jvm-dump ::obj) in file Jvm/jdump.scm:125
#4 (compiler) in file Engine/compiler.scm:322
#5 (engine) in file Engine/engine.scm:34
#6 (<exit:1030:Init/main.scm:45> ::pair) in file Init/main.scm:43
#7 (<exit:1017:Init/main.scm:39> ::pair) in file Init/main.scm:45
#8 (main ::pair) in file Init/main.scm:39
#9 (bigloo_main ::obj) in file Init/main.scm
#10 (main ::(array of java.lang.String)) in file Init/main.scm

-0:2* *Bugloo stack* All (11,63) (Bug Stack Mouse)-----
globals (::pair) = (#0=#global [module: foo] [import: export] [eva
env (::Jvm.env) = #lenv [declarations: ((getbytes (method jstring (
me (::symbol) = c_foo
super (::symbol) = j_procedure

-0:2* *Bugloo args* All (4,30) (Bug Args Mouse)-----
```

# Emacs Environment (1/2)

- Source buffers:
  - Breakpoints list
- Execution State:
  - Debuggee inspection



```
File Edit Options Buffers Tools Help
[760] Break Handler (wait)
[761] Finalizer (wait)
[762] Reference Handler (wait)
[1] main (runnable)[]

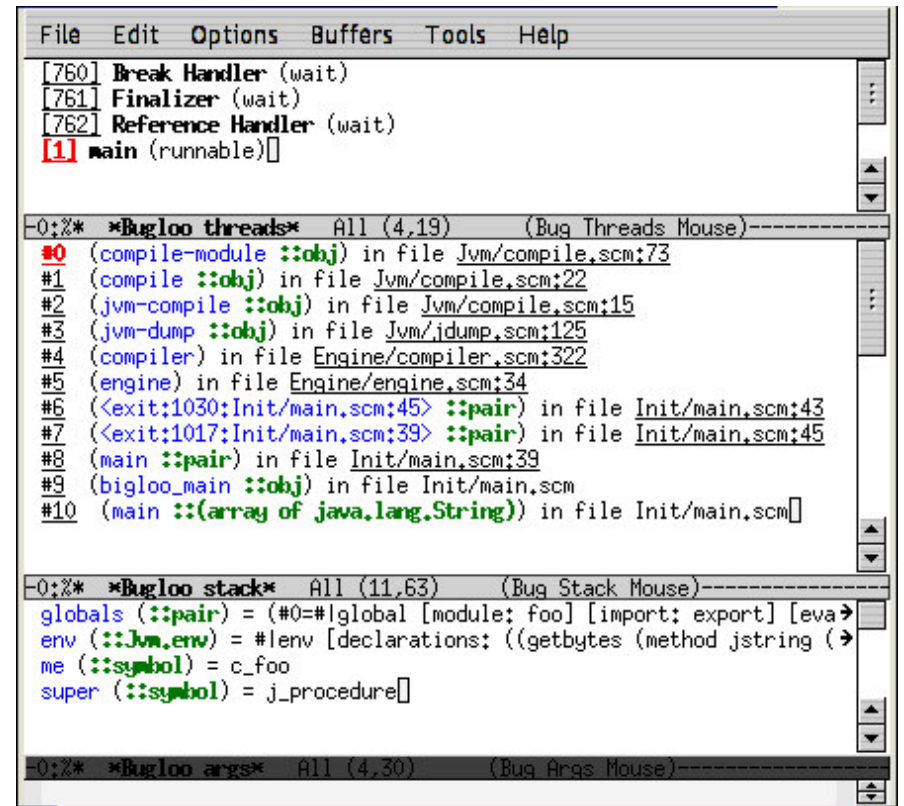
-0:2* *Bugloo threads* All (4,19) (Bug Threads Mouse)-----
#0 (compile-module ::obj) in file Jvm/compile.scm:73
#1 (compile ::obj) in file Jvm/compile.scm:22
#2 (jvm-compile ::obj) in file Jvm/compile.scm:15
#3 (jvm-dump ::obj) in file Jvm/jdump.scm:125
#4 (compiler) in file Engine/compiler.scm:322
#5 (engine) in file Engine/engine.scm:34
#6 (<exit:1030:Init/main.scm:45> ::pair) in file Init/main.scm:43
#7 (<exit:1017:Init/main.scm:39> ::pair) in file Init/main.scm:45
#8 (main ::pair) in file Init/main.scm:39
#9 (bigloo_main ::obj) in file Init/main.scm
#10 (main ::(array of java.lang.String)) in file Init/main.scm[]

-0:2* *Bugloo stack* All (11,63) (Bug Stack Mouse)-----
globals (::pair) = (#0=#global [module: foo] [import: export] [eva
env (::Jvm.env) = #lenv [declarations: ((getbytes (method jstring (
me (::symbol) = c_foo
super (::symbol) = j_procedure[]

-0:2* *Bugloo args* All (4,30) (Bug Args Mouse)-----
```

# Emacs Environment (1/2)

- Source buffers:
  - Breakpoints list
- Execution State:
  - Debuggee inspection
- Hyperlink configuration



The screenshot displays the Bugloo Emacs environment with several panels. The top panel shows the process list:

```
File Edit Options Buffers Tools Help
[760] Break Handler (wait)
[761] Finalizer (wait)
[762] Reference Handler (wait)
[1] main (runnable)
```

The middle panel, titled `*Bugloo threads*`, shows a list of threads:

```
-0:2* *Bugloo threads* All (4,19) (Bug Threads Mouse)-----
#0 (compile-module ::obj) in file Jvm/compile.scm:73
#1 (compile ::obj) in file Jvm/compile.scm:22
#2 (jvm-compile ::obj) in file Jvm/compile.scm:15
#3 (jvm-dump ::obj) in file Jvm/jdump.scm:125
#4 (compiler) in file Engine/compiler.scm:322
#5 (engine) in file Engine/engine.scm:34
#6 (<exit:1030:Init/main.scm:45> ::pair) in file Init/main.scm:43
#7 (<exit:1017:Init/main.scm:39> ::pair) in file Init/main.scm:45
#8 (main ::pair) in file Init/main.scm:39
#9 (bigloo_main ::obj) in file Init/main.scm
#10 (main ::(array of java.lang.String)) in file Init/main.scm
```

The bottom panel, titled `*Bugloo stack*`, shows the current stack frame:

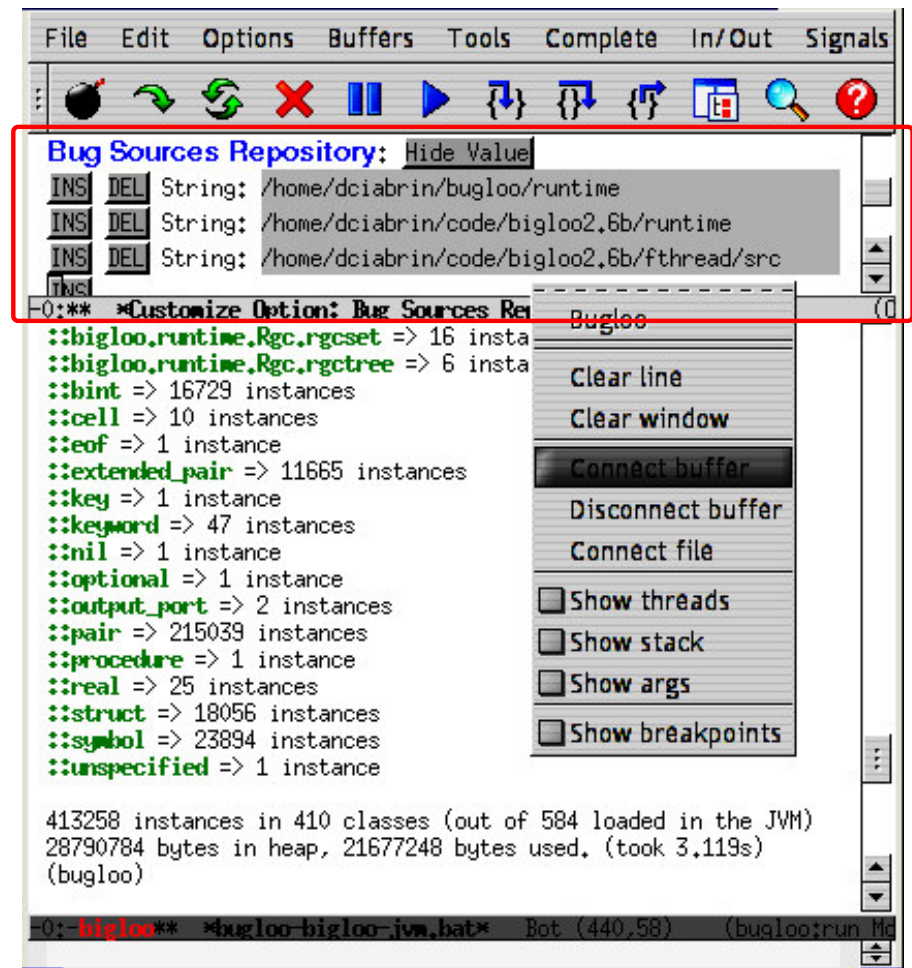
```
-0:2* *Bugloo stack* All (11,63) (Bug Stack Mouse)-----
globals (::pair) = (#0=#global [module: foo] [import: export] [eva
env (::Jvm.env) = #lenv [declarations: ((getbytes (method jstring (
me (::symbol) = c_foo
super (::symbol) = j_procedure
```

The bottom-most panel, titled `*Bugloo args*`, shows the arguments:

```
-0:2* *Bugloo args* All (4,30) (Bug Args Mouse)-----
```

# Emacs Environment (2/2)

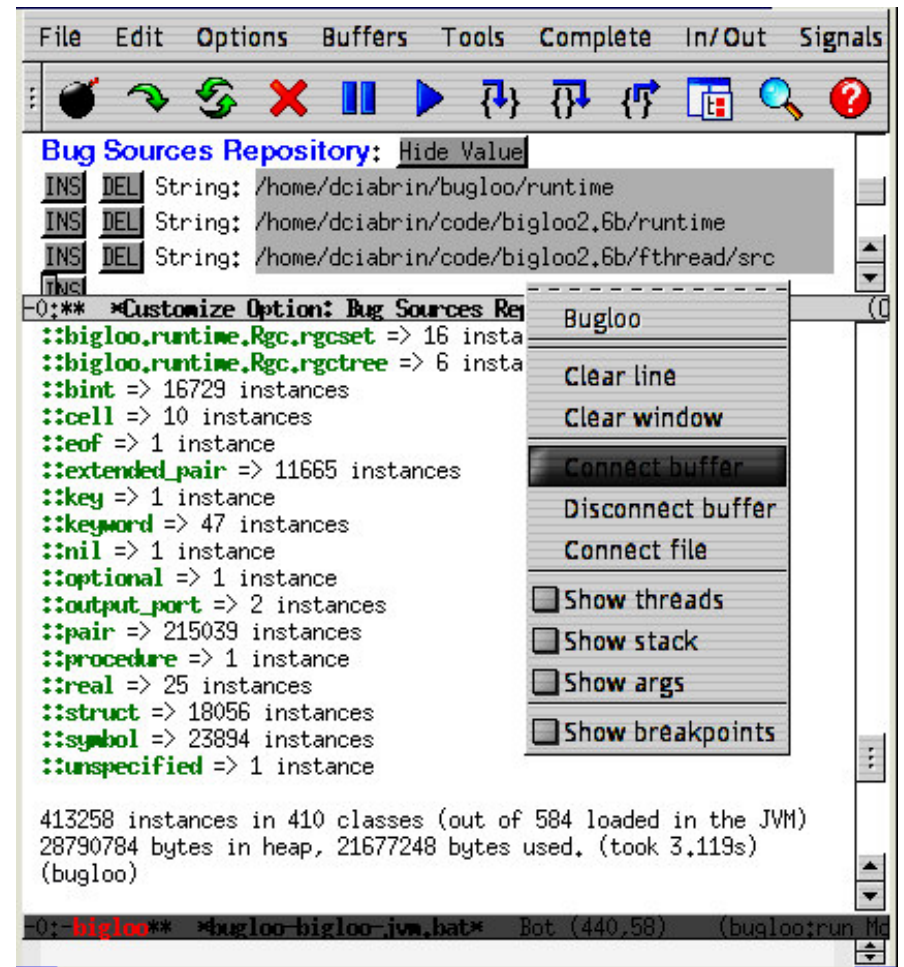
Source path repository:



# Emacs Environment (2/2)

## Source path repository:

- Where to look for sources
- Customization *à la* Emacs



The screenshot shows the Emacs interface with the Bug Sources Repository window open. The window title is "Bug Sources Repository: Hide Value". The menu bar includes File, Edit, Options, Buffers, Tools, Complete, In/Out, and Signals. The toolbar contains icons for various actions like opening, saving, and running. The main area displays a list of source paths:

```
INS DEL String: /home/dciabrin/bugloo/runtime
INS DEL String: /home/dciabrin/code/bigloo2.6b/runtime
INS DEL String: /home/dciabrin/code/bigloo2.6b/ftthread/src
```

A context menu is open over the first path, showing options: Bugloo, Clear line, Clear window, Connect buffer, Disconnect buffer, Connect file, Show threads, Show stack, Show args, and Show breakpoints. Below the list, the following summary is shown:

```
413258 instances in 410 classes (out of 584 loaded in the JVM)
28790784 bytes in heap, 21677248 bytes used. (took 3.119s)
(bugloo)
```

The status bar at the bottom shows: "0: bigloo\*\* \*bugloo-bigloo-jvm.bat\* Bot (440,58) (bugloorun Mc".

# Emacs Environment (2/2)



## Source path repository:

- Where to look for sources
- Customization *à la* Emacs

## Command line buffer:

```
File Edit Options Buffers Tools Complete In/Out Signals
Bug Sources Repository: Hide Value
INS DEL String: /home/dciabrin/bugloo/runtime
INS DEL String: /home/dciabrin/code/bigloo2.6b/runtime
INS DEL String: /home/dciabrin/code/bigloo2.6b/ftthread/src
-0:** *Customize Option: Bug Sources Re
::bigloo_runtime.Rgc.rgcset => 16 insta
::bigloo_runtime.Rgc.rgctree => 6 insta
::bint => 16729 instances
::cell => 10 instances
::eof => 1 instance
::extended_pair => 11665 instances
::key => 1 instance
::keyword => 47 instances
::nil => 1 instance
::optional => 1 instance
::output_port => 2 instances
::pair => 215039 instances
::procedure => 1 instance
::real => 25 instances
::struct => 18056 instances
::symbol => 23894 instances
::unspecified => 1 instance
413258 instances in 410 classes (out of 584 loaded in the JVM)
28790784 bytes in heap, 21677248 bytes used. (took 3.119s)
(bugloo)
-0: -bugloo** *bugloo-bigloo-jvm.bat* Bot (440,58) (bugloorun Mc
```





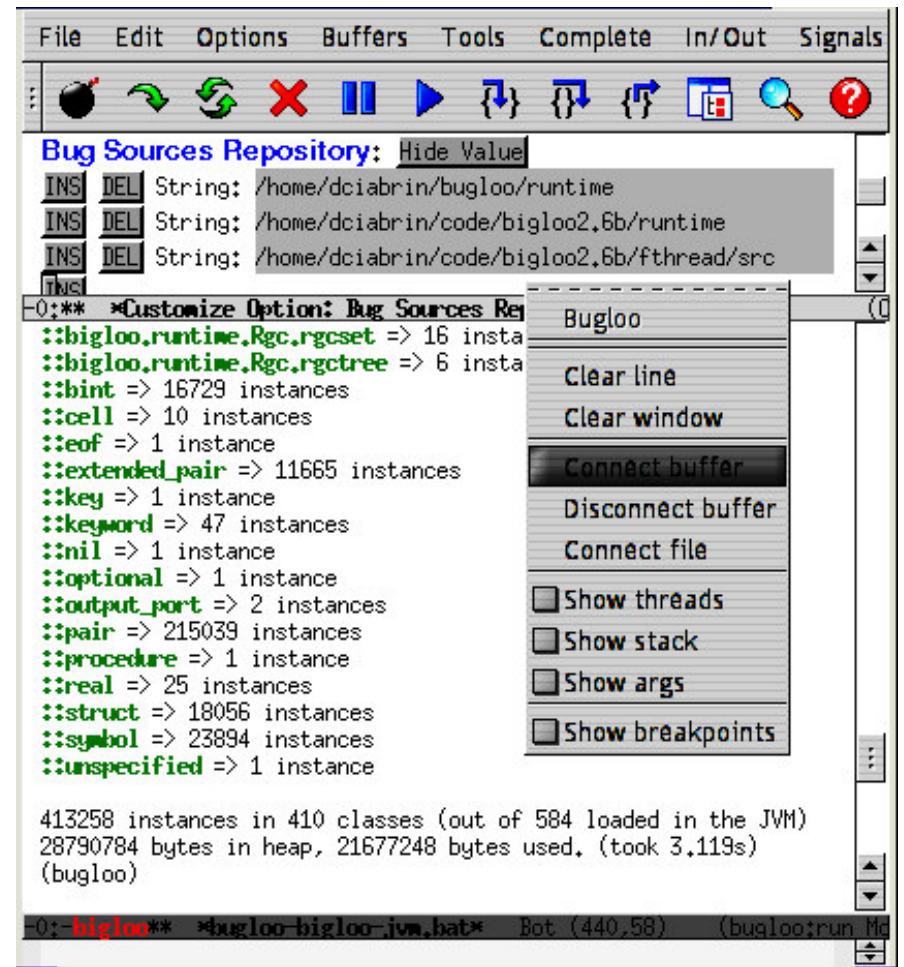
# Emacs Environment (2/2)

## Source path repository:

- Where to look for sources
- Customization *à la* Emacs

## Command line buffer:

- Manual interactions
- For advanced features



The screenshot shows the Emacs interface with the Bug Sources Repository window open. The window title is "Bug Sources Repository: Hide Value". It contains a list of source paths:

- INS DEL String: /home/dciabrin/bugloo/runtime
- INS DEL String: /home/dciabrin/code/bigloo2.6b/runtime
- INS DEL String: /home/dciabrin/code/bigloo2.6b/ftthread/src

A context menu is open over the list, showing options: Bugloo, Clear line, Clear window, Connect buffer, Disconnect buffer, Connect file, Show threads, Show stack, Show args, and Show breakpoints. The "Connect buffer" option is highlighted.

Below the list, the following text is displayed:

```
0:*** *Customize Option: Bug Sources Re
::bigloo_runtime.Rgc.rgcset => 16 insta
::bigloo_runtime.Rgc.rgctree => 6 insta
::bint => 16729 instances
::cell => 10 instances
::eof => 1 instance
::extended_pair => 11665 instances
::key => 1 instance
::keyword => 47 instances
::nil => 1 instance
::optional => 1 instance
::output_port => 2 instances
::pair => 215039 instances
::procedure => 1 instance
::real => 25 instances
::struct => 18056 instances
::symbol => 23894 instances
::unspecified => 1 instance

413258 instances in 410 classes (out of 584 loaded in the JVM)
28790784 bytes in heap, 21677248 bytes used. (took 3.119s)
(bugloo)
```

The bottom of the window shows the command line buffer with the text: "0: bigloo\*\* \*bugloo-bigloo-jvm.bat\* Bot (440,58) (bugloorun Mc

# Custom Debug Features



- Events recording
  - Trace of function calls
- Eval code at run-time
  - Memo-breakpoints
- Memory Debugging
  - Back references paths



# Event Recording



- All events that occur during a debug session:
  - History of user commands
    - Simple *replay* mechanism
  - Traces of debuggee events
    - Variable accesses, functions calls



# Event Recording - example

```
1: (define (go args)
2:   (my-map (lambda (x) (+ x 1)) '(1 2)))
3:
4: (define (my-map f l)
5:   (if (null? l)
6:       '()
7:       (cons (f (car l)) (my-map f (cdr l)))))
```

# Event Recording - example

```
1: (define (go args)
2:   (my-map (lambda (x) (+ x 1)) '(1 2)))
3:
4: (define (my-map f l)
5:   (if (null? l)
6:       '()
7:       (cons (f (car l)) (my-map f (cdr l)))))
```

```
(bugloo) (info stack)
```

```
#0 (my-map ::procedure ::obj) in file trace.scm:6
#1 (my-map ::procedure ::obj) in file trace.scm:7
#2 (my-map ::procedure ::obj) in file trace.scm:7
#3 (go ::pair) in file trace.scm:2
```

# Event Recording - example

```
1: (define (go args)
2:   (my-map (lambda (x) (+ x 1)) '(1 2)))
3:
4: (define (my-map f l)
5:   (if (null? l)
6:       '()
7:       (cons (f (car l)) (my-map f (cdr l)))))
```

```
(bugloo) (trace list)
```

```
. (go ::pair) in file trace.scm:2
. (my-map ::procedure ::obj) in file trace.scm:4
. (<lambda:2> ::obj) in file trace.scm:2
. (my-map ::procedure ::obj) in file trace.scm:4
. (<lambda:2> ::obj) in file trace.scm:2
. (my-map ::procedure ::obj) in file trace.scm:4
```

# Eval code at runtime



- Bugloo uses the built-in Scheme interpreter
  - Debugger: eval arbitrary S-exp
  - Debuggee: conditional breakpoints
- In Bugloo, a condition is a lambda
  - Various usage:
    - insert code without recompiling
    - a closure is a **memo-condition**



# Eval code at runtime - example

```
1: (define (mouse-click-handler e::int)
2:   (cond
3:     ((= e 1) (print "Button 1 pressed"))
4:     ((= e 2) (print "Button 2 pressed"))
5:     (else (print "never mind"))))
```



# Eval code at runtime - example

```
1: (define (mouse-click-handler e::int)
2:   (cond
3:     ((= e 1) (print "Button 1 pressed"))
4:     ((= e 2) (print "Button 2 pressed"))
5:     (else (print "never mind"))))
```

```
(let ((but2-ok #f))
  (lambda (env)
    (cond
      ((and (= (dbg env 'e) 1) but2-ok)
        (set! but2-ok #f) #t)
      ((= (dbg env 'e) 2)
        (set! but2-ok #t))))))
```

# Memory Debugging



- Scheme provides a GC
  - Not sufficient to avoid memory leaks !
- Services provided by Bugloo:
  - Heap inspector
    - To monitor memory consumption
  - Incoming references
    - To exhibit sharing properties
  - Back references path
    - Which GC root is responsible of a leak



# Memory Debugging - example

```
1: (module leak2
2:   (export (class ast-node
3:           type::symbol
4:           value::obj))
5:   (main compile))
6:
7: (define *nodes-cache* (make-hashtable))
8:
9: (define (compile args)
10:  (let ((obj (file->ast (car args))))
11:    (set! obj (ast->il obj))
12:    (set! obj (il->bytecode obj))
13:    (bytecode->file obj (cadr args))))
```

# Memory Debugging - example



(bugloo) (gc)



# Memory Debugging - example



```
(bugloo) (gc)
(bugloo) (info heap "::")
::ast-node => 29988 instances
::leak2 => 11 instances
::pair => 91109 instances
::struct => 1 instance
::bint => 25982 instances
::nil => 1 instance
::procedure => 1 instance
::symbol => 800 instances
::cell => 3 instances
::eof => 1 instance
::key => 1 instance
::nil => 1 instance
::unspecified => 1 instance
5137224 bytes used. (0.929s)
```



# Memory Debugging - example



```
(bugloo) (gc)
```

```
(bugloo) (heap get "::ast-node" 0)
```

```
(bugloo) (info heap "::")
```

```
::ast-node => 29988 instances
```

```
::leak2 => 11 instances
```

```
::pair => 91109 instances
```

```
::struct => 1 instance
```

```
::bint => 25982 instances
```

```
::nil => 1 instance
```

```
::procedure => 1 instance
```

```
::symbol => 800 instances
```

```
::cell => 3 instances
```

```
::eof => 1 instance
```

```
::key => 1 instance
```

```
::nil => 1 instance
```

```
::unspecified => 1 instance
```

```
5137224 bytes used. (0.929s)
```



# Memory Debugging - example



```
(bugloo) (gc)
(bugloo) (info heap "::")
::ast-node => 29988 instances
::leak2 => 11 instances
::pair => 91109 instances
::struct => 1 instance
::bint => 25982 instances
::nil => 1 instance
::procedure => 1 instance
::symbol => 800 instances
::cell => 3 instances
::eof => 1 instance
::key => 1 instance
::nil => 1 instance
::unspecified => 1 instance
5137224 bytes used. (0.929s)
```

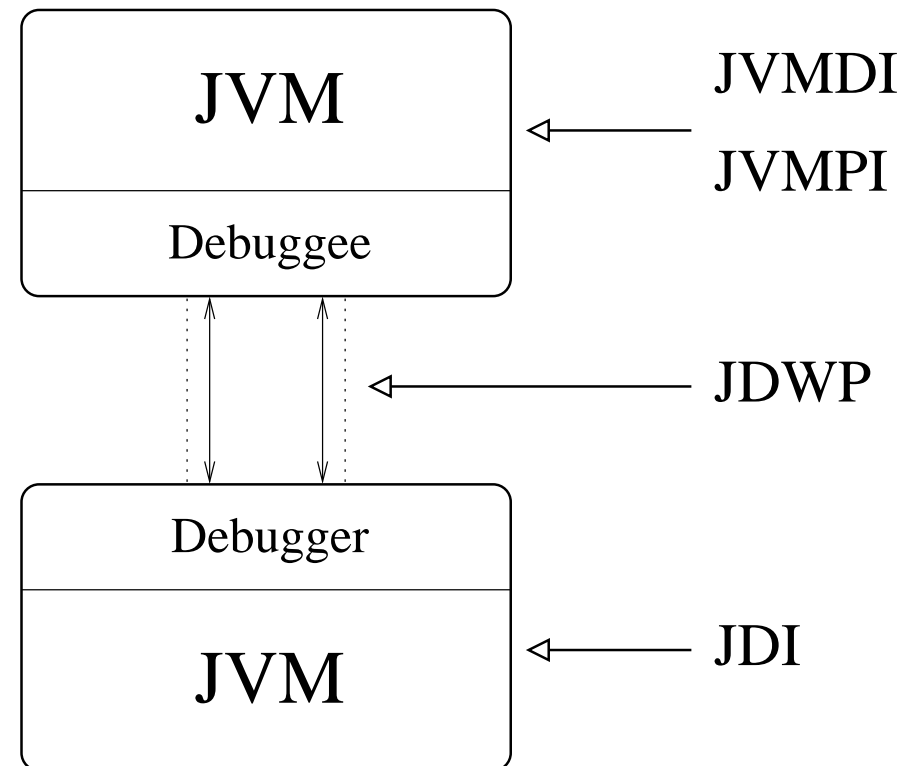
```
(bugloo) (heap get "::ast-node" 0)
(bugloo) (backref %obj%)
#0 ::ast-node
| field car
#1 ::pair
| field car
#2 ::pair
| at index 4082
#3 ::vector
| at index 2
#4 ::vector
| field values
#5 ::struct =====> *nodes-cache*
command took 0.743s.
```



# JVM Debugging Architecture



- Debugging with two JVMs
  - JVMDI: instrumentation
  - JDI: control (Java)
- *Event-driven* Communication
  - Manipulation of stubs
  - JDWP abstract channel
- Embed code in the debuggee
  - conditional breakpoints
  - memory debugging





# Mapping Scheme To JVM (1/2)



- Bigloo directly maps into JVM:
  - modules  $\Rightarrow$  classes
  - functions  $\Rightarrow$  methods



# Mapping Scheme To JVM (1/2)



- Bigloo directly maps into JVM:
  - modules  $\Rightarrow$  classes
  - functions  $\Rightarrow$  methods
- Proper display of Bigloo objects:



# Mapping Scheme To JVM (1/2)



- Bigloo directly maps into JVM:
  - modules  $\Rightarrow$  classes
  - functions  $\Rightarrow$  methods
- Proper display of Bigloo objects:

Normal display:

```
f (::procedure) = #<procedure:1>
```



# Mapping Scheme To JVM (1/2)



- Bigloo directly maps into JVM:
  - modules  $\Rightarrow$  classes
  - functions  $\Rightarrow$  methods
- Proper display of Bigloo objects:

Bugloo display:

```
f (::procedure) = procedure (foo ::obj) in file foo.scm:2
```



# Mapping Scheme To JVM (1/2)



- Bigloo directly maps into JVM:
  - modules  $\Rightarrow$  classes
  - functions  $\Rightarrow$  methods
- Proper display of Bigloo objects:

Bugloo display:

```
f (::procedure) = procedure (foo ::obj) in file foo.scm:2
```

- Some construction are emulated:
  - closure, higher order functions



# Mapping Scheme To JVM (2/2)



## Hide internals of Bigloo compilation

- Filtering Single Stepping:

- Step out of JVM constructors:

- ```
(filter ext add ("bigloo\\..*\\.<clinit>" . out))
```

- Don't stop in higher-order call dispatcher:

- ```
(filter ext add ("\\.funcall[0-4]\\(" . next))
```



# Mapping Scheme To JVM (2/2)



## Hide internals of Bigloo compilation

- Filtering Single Stepping:
  - Step out of JVM constructors:  

```
(filter ext add ("bigloo\\..*\\.<clinit>" . out))
```
  - Don't stop in higher-order call dispatcher:  

```
(filter ext add ("\\.funcall[0-4]\\(" . next))
```
- Limitations
  - Does not filter steps inside a function
  - Functions still visible in the stack frame



# Performances

---



- Performance penalties are limited:





# Performances



- Performance penalties are limited:
  - 1.5% to 6% slower than normal execution



# Performances



- Performance penalties are limited:
  - 1.5% to 6% slower than normal execution
  - No impact on memory consumption



# Performances



- Performance penalties are limited:
  - 1.5% to 6% slower than normal execution
  - No impact on memory consumption
- JIT stays enabled during debug



# Performances



- Performance penalties are limited:
  - 1.5% to 6% slower than normal execution
  - No impact on memory consumption
- JIT stays enabled during debug
- Good performances for memory debugging:
  - back-reference path (546 links): 4.5s on a 20 Mb heap (> 396000 objects) (Athlon XP 1900+)



# Performances



- Performance penalties are limited:
  - 1.5% to 6% slower than normal execution
  - No impact on memory consumption
- JIT stays enabled during debug
- Good performances for memory debugging:
  - back-reference path (546 links): 4.5s on a 20 Mb heap (> 396000 objects) (Athlon XP 1900+)
- Can debug the Bigloo compiler ( $\approx$  130000 lines)



# Conclusion

---



- We have developed Bugloo:
  - Can debug Bigloo programs compiled for JVM
  - Source level debugger + custom features
  - Integration into the (X)Emacs editor



# Conclusion



- We have developed Bugloo:
  - Can debug Bigloo programs compiled for JVM
  - Source level debugger + custom features
  - Integration into the (X)Emacs editor
- Advantages of the JVM as a debug platform:
  - Clean API to instrument the debuggee
  - Same classfile for debug and for performances
  - Usable for debugging large programs



# Conclusion



- We have developed Bugloo:
  - Can debug Bigloo programs compiled for JVM
  - Source level debugger + custom features
  - Integration into the (X)Emacs editor
- Advantages of the JVM as a debug platform:
  - Clean API to instrument the debuggee
  - Same classfile for debug and for performances
  - Usable for debugging large programs

<http://www-sop.inria.fr/mimosa/fp/Bugloo>

