Ruby - Bug #15980

Coverage shows while/until after raise if/unless as uncovered line

07/03/2019 03:48 AM - jeremyevans0 (Jeremy Evans)

Status:	Closed		
Priority:	Normal		
Assignee:	mame (Yusuke Endoh)		
Target version:			
ruby -v:	ruby 2.7.0dev (2019-07-03) [x86_64-openbsd6.5]	**Backport:**	2.4: UNKNOWN, 2.5: UNKNOWN, 2.6: DONE

Description

The following code shows line 2 (while true) as uncovered:

```
raise if 1 == 2
while true
  break
end
```

Coverage reports the following for this file: [1, 0, 1, nil]. Note that true isn't important, any while condition will work. However, if you change line 1 to raise if false, line 1 shows nil coverage, and line 2 shows as covered ([nil, 1, 1, nil]). That leads me to believe this issue is related to the optimizer.

I bisected this to 100bf2757468439106775a7d95a791a8c10b874a, which certainly appears related.

This is not a theoretical case, it affected three lines in Sequel. While not a major problem, I do think a fix should be backported to 2.6.

Note that this only affects line coverage. Branch coverage shows:

```
{"file.rb"=>
  {:branches=>
    {[:if, 0, 1, 0, 1, 15]=>
      {[:then, 1, 1, 0, 1, 5]=>0, [:else, 2, 1, 0, 1, 15]=>1},
     [:while, 3, 2, 0, 4, 3]=>{[:body, 4, 3, 2, 3, 7]=>1}}}}
```

If you run with both branch and line coverage, line coverage shows correctly.

This affects while/until after a line with raise ... if ... or raise ... unless .... If you switch to if ...; raise ...; end, then line coverage shows correctly.

Related issues:

Has duplicate Ruby - Bug #16397: Line coverage is broken for until and while ...	**Closed**

Associated revisions

Revision f9e5c74cd24025a5aa19e318e8fecabf207f1b7b - 12/04/2019 01:40 AM - mame (Yusuke Endoh)

compile.c: stop wrong peephole optimization when covearge is enabled

jump-jump optimization ignores the event flags of the jump instruction
being skipped, which leads to overlook of line events.

This changeset stops the wrong optimization when coverage measurement is
neabled and when the jump instruction has any event flag.

Note that this issue is not only for coverage but also for TracePoint,
and this change does not fix TracePoint.
However, fixing it fundamentally is tough (which requires revamp of
the compiler). This issue is critical in terms of coverage measurement,
but minor for TracePoint (ko1 said), so we here choose a stopgap
measurement.

[Bug #15980] [Bug #16397]

Note for backporters: this changeset can be viewed by git diff -w.

Revision f9e5c74cd24025a5aa19e318e8fecabf207f1b7b - 12/04/2019 01:40 AM - mame (Yusuke Endoh)

compile.c: stop wrong peephole optimization when covearge is enabled

jump-jump optimization ignores the event flags of the jump instruction
being skipped, which leads to overlook of line events.

This changeset stops the wrong optimization when coverage measurement is
neabled and when the jump instruction has any event flag.

Note that this issue is not only for coverage but also for TracePoint,
and this change does not fix TracePoint.
However, fixing it fundamentally is tough (which requires revamp of
the compiler). This issue is critical in terms of coverage measurement,
but minor for TracePoint (ko1 said), so we here choose a stopgap
measurement.

[Bug #15980] [Bug #16397]

Note for backporters: this changeset can be viewed by git diff -w.

Revision f9e5c74c - 12/04/2019 01:40 AM - mame (Yusuke Endoh)

compile.c: stop wrong peephole optimization when covearge is enabled

jump-jump optimization ignores the event flags of the jump instruction
being skipped, which leads to overlook of line events.

This changeset stops the wrong optimization when coverage measurement is
neabled and when the jump instruction has any event flag.

Note that this issue is not only for coverage but also for TracePoint,
and this change does not fix TracePoint.
However, fixing it fundamentally is tough (which requires revamp of
the compiler). This issue is critical in terms of coverage measurement,
but minor for TracePoint (ko1 said), so we here choose a stopgap
measurement.

[Bug #15980] [Bug #16397]

Note for backporters: this changeset can be viewed by git diff -w.

Revision 3f4f5fdf0b0ef9df7a8a32a73b4af2c46a2373a5 - 02/28/2021 02:16 PM - U.Nakamura

merge revision(s) f9e5c74c: [Backport #15980]

```
    compile.c: stop wrong peephole optimization when covearge is enabled

    jump-jump optimization ignores the event flags of the jump instruction
    being skipped, which leads to overlook of line events.

    This changeset stops the wrong optimization when coverage measurement is
    neabled and when the jump instruction has any event flag.

    Note that this issue is not only for coverage but also for TracePoint,
    and this change does not fix TracePoint.
    However, fixing it fundamentally is tough (which requires revamp of
    the compiler).  This issue is critical in terms of coverage measurement,
    but minor for TracePoint (ko1 said), so we here choose a stopgap
    measurement.

    [Bug #15980] [Bug #16397]

    Note for backporters: this changeset can be viewed by `git diff -w`.
    ---
     compile.c                      | 232 ++++++++++++++++++++++++++------------------
     test/coverage/test_coverage.rb |  12 +++
     2 files changed, 141 insertions(+), 103 deletions(-)
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_6@67898 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 3f4f5fdf0b0ef9df7a8a32a73b4af2c46a2373a5 - 02/28/2021 02:16 PM - U.Nakamura

merge revision(s) f9e5c74c: [Backport #15980]

```
    compile.c: stop wrong peephole optimization when covearge is enabled
```

```
    jump-jump optimization ignores the event flags of the jump instruction
    being skipped, which leads to overlook of line events.

    This changeset stops the wrong optimization when coverage measurement is
    neabled and when the jump instruction has any event flag.

    Note that this issue is not only for coverage but also for TracePoint,
    and this change does not fix TracePoint.
    However, fixing it fundamentally is tough (which requires revamp of
    the compiler).  This issue is critical in terms of coverage measurement,
    but minor for TracePoint (ko1 said), so we here choose a stopgap
    measurement.

    [Bug #15980] [Bug #16397]

    Note for backporters: this changeset can be viewed by `git diff -w`.
    ---
     compile.c                     | 232 ++++++++++++++++++++++++------------------
     test/coverage/test_coverage.rb |  12 +++
     2 files changed, 141 insertions(+), 103 deletions(-)
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_6@67898 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 3f4f5fdf - 02/28/2021 02:16 PM - U.Nakamura

merge revision(s) f9e5c74c: [Backport #15980]

```
    compile.c: stop wrong peephole optimization when coveargе is enabled

    jump-jump optimization ignores the event flags of the jump instruction
    being skipped, which leads to overlook of line events.

    This changeset stops the wrong optimization when coverage measurement is
    neabled and when the jump instruction has any event flag.

    Note that this issue is not only for coverage but also for TracePoint,
    and this change does not fix TracePoint.
    However, fixing it fundamentally is tough (which requires revamp of
    the compiler).  This issue is critical in terms of coverage measurement,
    but minor for TracePoint (ko1 said), so we here choose a stopgap
    measurement.

    [Bug #15980] [Bug #16397]

    Note for backporters: this changeset can be viewed by `git diff -w`.
    ---
     compile.c                     | 232 ++++++++++++++++++++++++------------------
     test/coverage/test_coverage.rb |  12 +++
     2 files changed, 141 insertions(+), 103 deletions(-)
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_6@67898 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 12/03/2019 09:52 PM - jeremyevans0 (Jeremy Evans)

- Has duplicate Bug #16397: Line coverage is broken for until and while after guard clause added

#2 - 12/03/2019 10:05 PM - puchuu (Andrew Aladjev)

Hello. This thing affects both return, break and raise guard clauses. I want to insist on covering this issue with tests, test_line_coverage_for_multiple_lines is too weak. I will try to prepare tests.

Unfortunately this bug was not fixed by 100bf2757468439106775a7d95a791a8c10b874a. I am able to reproduce it now using ruby-head with the following code:

test.rb:

```
require "coverage"

Coverage.start

require_relative "cov"

pp Coverage.result
```

cov.rb:

```
a = 2

raise StandardError if a.zero?

a -= 1 until a.zero?

puts a
```

It returns [1, nil, 1, nil, 0, nil, 1], but the right answer is [1, nil, 1, nil, 1, nil, 1].

ruby -v equals ruby 2.7.0dev (2019-12-03T21:40:54Z trunk 8852fa8760) [x86_64-linux]

#3 - 12/03/2019 10:32 PM - mame (Yusuke Endoh)

Oops, I overlooked this issue. Sorry.

This is a tough problem. The compiler produces the following code:

```
0006 branchunless 13
...
0013 jump 17          (Line 2)
...
0017 putnil           (Line 3)
```

The peephole optimizer changes it to:

```
0006 branchunless 17
...
0013 jump 17          (Line 2)
...
0017 putnil           (Line 3)
```

Now "jump 17" instruction is unreachable, so Line 2 event is never fired. But coverage measurement doesn't know it, so it initializes zero entry for Line 2 in coverage.

I think of some options to fix:

1. Stop the peephole optimization (in some cases); it reduces performance even when coverage is not used, so I don't want to do this.
2. Remove unreachable instructions; it requires revamp of the compiler.
3. Analyze unreachable instructions and suppress coverage entry; it is relatively easy, but still requires unreachable code analysis.

@ko1 (Koichi Sasada) Do you have any idea?

#4 - 12/03/2019 10:44 PM - jeremyevans0 (Jeremy Evans)

mame (Yusuke Endoh) wrote:

> I think of some options to fix:
>
> 1. Stop the peephole optimization (in some cases); it reduces performance even when coverage is not used, so I don't want to do this.
> 2. Remove unreachable instructions; it requires revamp of the compiler.
> 3. Analyze unreachable instructions and suppress coverage entry; it is relatively easy, but still requires unreachable code analysis.

My idea for a hacky workaround: As the correct result is shown when branch coverage is enabled, always run with branch coverage enabled, even if the branch results will not be used.

#5 - 12/04/2019 12:53 AM - mame (Yusuke Endoh)

jeremyevans0 (Jeremy Evans) wrote:

> My idea for a hacky workaround: As the correct result is shown when branch coverage is enabled, always run with branch coverage enabled, even if the branch results will not be used.

This issue is not only coverage but also TracePoint line events. It does not fix TracePoint.

I was trying to implement Approach 3, but I found it does not fix the issue. The instruction jump 17 that I said in my previous comment is theoretically reachable if 1==2 returns true and raise is redefined as no-op.

Here is a patch by Approach 1:

```
diff --git a/compile.c b/compile.c
```

```
index 0b808342c0..150515139c 100644
--- a/compile.c
+++ b/compile.c
@@ -2861,6 +2861,7 @@ iseq_peephole_optimize(rb_iseq_t *iseq, LINK_ELEMENT *list, const int do_tailcal
          *   if L2
          */
         INSN *nobj = (INSN *)get_destination_insn(iobj);
+        if (nobj->insn_info.events == 0) {
             INSN *pobj = (INSN *)iobj->link.prev;
             int prev_dup = 0;
             if (pobj) {
@@ -2965,6 +2966,7 @@ iseq_peephole_optimize(rb_iseq_t *iseq, LINK_ELEMENT *list, const int do_tailcal
                 nobj = (INSN *)get_destination_insn(nobj);
             }
         }
+    }
     if (IS_INSN_ID(iobj, pop)) {
         /*
```

It is so simple; it just suppresses the jump-jump peephole optimization when the jump instruction being skipped has an event flag. But I'm unsure how much it reduces performance.

#6 - 12/04/2019 01:19 AM - mame (Yusuke Endoh)

I talked with @ko1 (Koichi Sasada), and decide to introduce a stopgap measurement: stop the optimization in question only when coverage measurement is enabled. This is because this issue is critical in coverage measurement, but minor in TracePoint.

The current compiler has many minor flaws, so we need to revamp it anyway in near future.

I will commit a patch soon.

#7 - 12/04/2019 01:41 AM - mame (Yusuke Endoh)

- Status changed from Assigned to Closed

Applied in changeset git|f9e5c74cd24025a5aa19e318e8fecabf207f1b7b.

---

compile.c: stop wrong peephole optimization when covearge is enabled

jump-jump optimization ignores the event flags of the jump instruction
being skipped, which leads to overlook of line events.

This changeset stops the wrong optimization when coverage measurement is
neabled and when the jump instruction has any event flag.

Note that this issue is not only for coverage but also for TracePoint,
and this change does not fix TracePoint.
However, fixing it fundamentally is tough (which requires revamp of
the compiler). This issue is critical in terms of coverage measurement,
but minor for TracePoint (ko1 said), so we here choose a stopgap
measurement.

[Bug #15980] [Bug #16397]

Note for backporters: this changeset can be viewed by git diff -w.

#8 - 12/04/2019 09:26 AM - puchuu (Andrew Aladjev)

There is a workaround for everyone wants to use lines coverage only with 2.6.5 version (recommended by Jeremy Evans):

```
# Workaround for https://bugs.ruby-lang.org/issues/15980
require "coverage"

Coverage.module_eval do
  singleton_class.send :alias_method, :original_start, :start
  def self.start
    original_start :lines => true, :branches => true
  end

  singleton_class.send :alias_method, :original_result, :result
  def self.result
    original_result.transform_values { |coverage| coverage[:lines] }
  end
end
```

It works for simplecov.

#9 - 12/04/2019 04:09 PM - puchuu (Andrew Aladjev)

Yusuke Endoh, I've tested using the following code:

```
loop do
  break if rand < 0
  break while true

  next if rand < 0
  break until false

  return if rand < 0
  break while true

  raise if rand < 0
  break until false

  exit if rand < 0
  break while true

  break
end
```

Result is [1, 1, 1, nil, 1, 1, nil, 1, 1, nil, 1, 1, nil, 1, 1, nil, 1, nil], works fine, thank you.

#10 - 02/28/2021 02:16 PM - usa (Usaku NAKAMURA)

- Backport changed from 2.4: UNKNOWN, 2.5: UNKNOWN, 2.6: REQUIRED to 2.4: UNKNOWN, 2.5: UNKNOWN, 2.6: DONE

ruby_2_6 r67898 merged revision(s) f9e5c74c.