Ruby master - Feature #5588
add negation flag (v) to Regexp

11/08/2011 10:59 AM - sunaku (Suraj Kurapati)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>naruse (Yui NARUSE)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.6</td>
</tr>
</tbody>
</table>

**Description**

Please add a negation flag (v) to regexps which inverts them:

```
"ruby" =~ /perl/v       #=> true   (turn on negation)
"ruby" !~ /perl/v       #=> false  (turn on negation)
"ruby" =~ /(?v:perl)/   #=> true   (turn on negation)
"ruby" !~ /(?v:perl)/   #=> false  (turn on negation)
"ruby" =~ /(?-v:perl)/  #=> false  (turn off negation)
"ruby" !~ /(?-v:perl)/  #=> true   (turn off negation)
```

The flag name "v" comes from the ex(1) command of the same name.

This has beneficial applications where it is sometimes difficult to construct what you want to match but much easier to construct what you do not want to match. Having this negation built in the regexp itself would remove the need for us to change our Ruby code to process a regexp in a different way.

For example, suppose that you are passing a regexp to the --name command-line option of MiniTest. This regexp tells MiniTest to only run those tests whose names match. If Ruby had a negation flag on its regexps, then it would be suddenly trivial to make MiniTest skip certain tests by negating the regexp we pass in.

In this manner, we get a beneficial feature without ever changing our Ruby code (the codebase of MiniTest in this example). :-)

Thanks for your consideration.

**Related issues:**

- Related to Ruby master - Feature #11049: Enumerable#grep_v (inversed grep) Closed

**History**

#1 - 11/08/2011 11:04 AM - sunaku (Suraj Kurapati)

Note that I am not asking for the ability to take an arbitrary regexp and construct its negated form (that is a hard problem). Instead, I am asking for a flag that simply inverts how a regexp is treated by the =~ and !~ operators (this is not a hard problem). The negation flag should not change the Regexp#source of a regexp.

Thanks for your consideration.

#2 - 11/08/2011 11:53 AM - rkh (Konstantin Haase)

It is not to hard to negate a regexp, though. Simply wrap it as negative look-ahead.

Konstantin

On Nov 7, 2011, at 23:04 , Suraj Kurapati wrote:

Issue #5588 has been updated by Suraj Kurapati.

Note that I am not asking for the ability to take an arbitrary regexp and construct its negated form (that is a hard problem). Instead, I am asking for a flag that simply inverts how a regexp is treated by the

#3 - 11/08/2011 01:54 PM - sunaku (Suraj Kurapati)
Good idea, but negative lookahead isn't the same as negation:

```ruby
ruby 1.9.3p0 (2011-10-30 revision 33570) [x86_64-linux]

"rubyperl" =~ /(? perl)/
  0
"rubyperl" =~ /perl/
  4
"rubyperl" !~ /perl/
false
```

Thanks for your consideration.

---

**#4 - 11/08/2011 02:11 PM - matz (Yukihiro Matsumoto)**

How do you treat positional information with v flag?

---

**#5 - 11/08/2011 03:59 PM - alexeymuranov (Alexey Muranov)**

I would suggest to deal with it a new (sub)class instead: arbitrary boolean combinations of regexps, without regard for positional information.

---

**#6 - 11/08/2011 06:32 PM - sunaku (Suraj Kurapati)**

Hello Matz,

What do you mean by positional information:

- anchors (\A, $, \G, etc.) ?
- capture-group numbers ($1, $2, $3, etc.) ?
- MatchData#begin, #end, #offset ？

The v flag should only affect the low-level =~ and !~ operators in the C implementation. All subsequent processing should be performed as normal.

Thanks for your consideration.

---

**#7 - 11/09/2011 08:32 AM - sunaku (Suraj Kurapati)**

I tried to implement this feature in this patch:

https://github.com/sunaku/ruby/commit/79305ba55c7ece5501c9219942eaf30e01a370a9

I was able to make Ruby recognize 'v' as an embedded regexp flag, and was able to create regexps via the new `Regexp::NEGATED` constant.

However, I am stuck on the following things. Any tips? :-)

- We can pass '(?v:) in embedded regexp but it does not take effect. The resulting regexp object's #options field does not reflect 'v'.
- Make the parser accept 'v' as option at the end of literal regexps.

Thanks for your consideration.

---


2011/11/9 Suraj Kurapati sunaku@gmail.com:

- We can pass '(?v:) in embedded regexp but it does not take effect. The resulting regexp object's #options field does not reflect 'v'.

The option can be embedded to middle of a regexp: /foo(?v:bar)baz/

I think it doesn't work.

--

Tanaka Akira

---

**#9 - 11/09/2011 06:38 PM - sunaku (Suraj Kurapati)**

You are correct, Tanaka. I have been exploring the regexp implementation to understand why and I learned that the negate flag must be compiled as an opcode (similar to multiline and ignorecase) into the re_pattern_buffer->p string.
My current approach is to compile (?v:...) into two opcodes (OP_BEGIN_NEGATE and OP_END_NEGATE) with normal compilation of the ... stuff inside. Later, when match_at() in regexec.c:1254 is processing the compiled regexp and encounters OP_END_NEGATE, it will perform the negation:

If we consumed > 0 input characters since OP_BEGIN_NEGATE, then we have successfully matched the ... stuff inside the original (?v:...) regexp. So we need to stop further processing by returning ONIG_MISMATCH.

Otherwise, we did not consume any input characters since OP_BEGIN_NEGATE, so we continue processing by returning ONIG_NORMAL. This effectively treats the (?v:...) as a zero-length regexp, which always matches of course.

That's my plan for now. Any comments?

Thanks for your consideration.

#10 - 11/10/2011 09:46 AM - sunaku (Suraj Kurapati)
After several deep excursions into the regexp codebase, I've had enough. :)

As Tanaka(!) showed in 2007, negative global regexps are already there:

http://www.ruby-forum.com/topic/133413#595368

"rubypert" =~ /(?!(?:perl).)+$/

nil

So you can close this feature request now. Sorry for the noise.

#11 - 11/10/2011 10:32 AM - sunaku (Suraj Kurapati)
Alas, I was unable to resist the lure of implementing this, so I'm back to give this another try.

My current approach is to expand (?v:STUFF) into (?:(?!STUFF).) when the regexp AST is built.

The goal is to have an negated embedded regexp say: "there is something here that is NOT this".

#12 - 11/11/2011 08:38 AM - sunaku (Suraj Kurapati)
I did it! :)

Please take a look:

https://github.com/sunaku/ruby/compare/5588_regexp_v

There are a few issues remaining with the implementation:

- Store snegate on STACK support nested embedded negated regexps.
- Find a better way to double-pop the stack in OP_NEGATE_END handler in regexec.c; currently it puts undefined values into s, p, etc. when the stack is popped for the second time in "goto fail" handler.

After that, I need to make Ruby parser accept /.../v as a literal regexp flag.

Thanks for your consideration.

#13 - 11/11/2011 09:00 AM - sunaku (Suraj Kurapati)
Allow me to explain the current embedded negated regexp implementation.

When parsing an embedded negated regexp (?v:r), we expand them into this:

OP_NEGATE_START(?r)?OP_NEGATE_END.*?

Here, OP_NEGATE_START and OP_NEGATE_END are opcodes in compiled pattern.

When the regexp engine (see match_at() in regexec.c) reaches OP_NEGATE_START, we store the current state of input (up to which character of the input string have we consumed/matched so far?) in the "snegate" variable.

The regexp engine then continues onward and eventually reaches OP_NEGATE_END. At this point, I compare the current state of input with "snegate". This tells us if the original embedded negated regexp (?v:r) has matched anything. Now we perform the negation:

If (?v:r) matched something, then treat this as a mismatch (prevent backtrack and goto fail). Otherwise, continue processing (the ".*?" after OP_NEGATE_END will take care of consuming any non-matching characters so that we can still proceed).

assert_no_match(/a(?v:b)c/, "abc")
assert_match(/a(?v:b)c/, "axc")
assert_match(/a(?v:b)c/, "ac")
assert_match(/a(?v:b)c/, "axbc")

I hope this helps you understand my approach. Please correct me if I made a mistake.

Thanks for your consideration.

#14 - 11/12/2011 05:31 PM - sunaku (Suraj Kurapati)
I have updated my patch to emit a single OP_NEGATE opcode after the negated embedded regexp (?v:...). This opcode double-pops the stack to prevent the optional alternation from succeeding. Please take a look:

https://github.com/sunaku/ruby/compare/5588_regexp_v

There might be a bug in the "(?v:r)" to "(?:rN)?" expansion (where "N" is OP_NEGATE) because DONIG_DEBUG_PARSE_TREE shows the expanded "(?:rN)?" twice:

PATTERN: /a(?v:b)c/ (ASCII-8BIT)
list:965430
string:96af40
enclose:965570
option:4096
quantifier:965520
{0,1}
string:965480

<quantifier:965520>{0,1} === BUG? Why is this here twice?
<string:965480>b

quantifier:96c830{0,-1}?
anychar:96ae00
string:96c920

c

I will iron out these issues and finish the implementation in due time.

Cheers.

#15 - 11/12/2011 06:26 PM - sunaku (Suraj Kurapati)
The double-printing of ENCLOSE_OPTION node was a bug in Oniguruma 5.9.2 and not in my code, for once! ;) I have submitted a fix for that bug to Kosako, the author of Oniguruma, accordingly. In case you are interested, here is the bug fix:

https://github.com/sunaku/ruby/commit/125c31a0fe42fb2937ea64c2f31293b81bb32d8b

Cheers.

#16 - 11/13/2011 06:59 PM - sunaku (Suraj Kurapati)
- File 5588_regexp_v.patch added

I fixed the 'v' flag parsing in literal regexps: the problem was the value of ONIG_OPTION_NEGATE that I chose (0x1000) collided with RB_ENCODING_OPTION mask (0xFF00).

Now the implementation is finally finished. Please review the attached patch and tell me what you think. If you like it, I will update the RDoc of the Regexp functions to reflect the new V flag.

Thanks for your consideration.

#17 - 11/14/2011 06:27 AM - sunaku (Suraj Kurapati)
I beautified my patches so they are easier to understand (especially the additions to test-regexp.rb in Ruby's test suite) and extracted the Oniguruma-only portions into a separate patch so that you can see what changes affect Ruby vs. Oniguruma. Here are the patches:

- Negated regexps in Oniguruma 5.9.2 (already submitted to Kosako):
  https://github.com/sunaku/onig-5.9.2/compare/v5.9.2...master

- Negated regexps in Oniguruma 5.9.2 plus integration in Ruby trunk:
  https://github.com/sunaku/ruby/compare/5588_regexp_v

I would like your feedback on these. Thanks for your consideration.

#18 - 11/14/2011 10:20 AM - sunaku (Suraj Kurapati)
I have explained this implementation in more detail on my blog:

http://snk.tuxfamily.org/log/oniguruma-negated-regexps.html
I hope that helps. Cheers :) 

---

#19 - 11/15/2011 04:55 AM - sunaku (Suraj Kurapati)
- File 0001-http-redmine.ruby-lang.org-issues-5588.patch added

#20 - 11/15/2011 09:45 AM - naruse (Yui NARUSE)
With your [ruby-core:41040]'s patch, I got following result.
Is this an expected result?

```
irb(main):029:0> /a(?v:b)c/=~"abc"
=> nil
irb(main):030:0> /a(?v:b)c/=~"ab_c"
=> nil
irb(main):031:0> /a(?v:b)c/=~"a_bc"
=> 0
```

#21 - 11/15/2011 03:17 PM - sunaku (Suraj Kurapati)
Hi Naruse,

Thanks for trying my patch and for your questions! :) 

Yui NARUSE wrote:

> With your [ruby-core:41040]'s patch, I got following result.
> Is this an expected result?

Yes, please allow me to explain why:

```
irb(main):029:0> /a(?v:b)c/=~"abc"
=> nil
```

First /a/ matched "a", then /(?v:b)/ mismatched "b". Failure.

```
irb(main):030:0> /a(?v:b)c/=~"ab_c"
=> nil
```

First /a/ matched "a", then /(?v:b)/ mismatched "b". Failure.

```
irb(main):031:0> /a(?v:b)c/=~"a_bc"
=> 0
```

First /a/ matched "a", then /(?v:b)/ matched ".", then /.?
(created when /(?v:b)/ was expanded into /(?:bN)?/) where N is
OP_NEGATE) matched "b", and finally /c/ matched "c". Success.

See also my explanation of partly negated regexp expansion:
http://snk.tuxfamily.org/log/oniguruma-negated-regexps.html

Thanks for your consideration.

#22 - 11/15/2011 03:27 PM - sunaku (Suraj Kurapati)

Suraj Kurapati wrote:

Yui NARUSE wrote:

> irb(main):031:0> /a(?v:b)c/=~"a_bc"
> => 0

then /(?v:b)/ matched "_"

Hmm, that explanation isn't fully accurate. Let me try again:

First /a/ matched "a", then /(?v:b)?\.*?\? (which is the parse-tree
expansion of /(?v:b)/) matched ".", and finally /c/ matched "c".
Success.
Sorry for the confusion.

#23 - 11/15/2011 06:08 PM - naruse (Yui NARUSE)
- Category set to core
- Status changed from Open to Assigned
- Assignee set to naruse (Yui NARUSE)
- Target version set to 2.0.0

Thank you for detailed explanation,
And sorry, [ruby-core:40932] explains it.
I believe this spec is wrong; (?v:foo) should match a sequence which doesn't include "foo".
OR a sequence just match "foo".
For example, /"(?v:<|&|")"/ AttValue of XML [http://www.w3.org/TR/xml/#NT-AttValue]
The behavior of this regexp seems /"[^<\&\"]+"/ but

irb(main):007:0> /"[^<\&\"]+"/ =~ '"aa<&a"'
=> nil
irb(main):008:0> /"(?v:<|&|")"/ =~ '"aa<&a"'
=> 0

This feels strange.

Do you have any use case which show current behavior is more reasonable?

#24 - 11/16/2011 04:46 AM - sunaku (Suraj Kurapati)
I don't have a use case for the current behavior; it was just the simplest way to remove non-matching input characters that obstructed the matching engine. And I agree with your examples; people would naturally think of /(?v:...)/ as a glorified form of /\(...)/.

The solution is to change the parse-tree expansion into this:

/ (?v:r) / => / (?:(?!rN).)/

In this manner, the /(?sN)/ acts as a barrier that only allows input characters that do not match r to be matched by the /. /

However, this seems very similar to Tanaka's 2007 solution:

/ (?v:r) / => / (?!(r)).)/

I will play with Oniguruma in GDB some more to learn how (!?) works.
Perhaps my OP_NEGATE modification is actually unnecessary. Cheers.

#25 - 11/16/2011 09:31 AM - sunaku (Suraj Kurapati)
Hello Naruse,

I have updated my patch to expand /(?v:r)/ into /(?!(rN))./:
[https://github.com/sunaku/ruby/compare/5588_regexp_v](https://github.com/sunaku/ruby/compare/5588_regexp_v)

It seems that OP_NEGATE is necessary after all, because without it, Oniguruma will try to match a non-anchored partly negated regexp to the rest of the input string.

For example, when processing /(?v:ruby)/ =~ "ruby", Oniguruma does:

1. /(?v:ruby)/ =~ "ruby" # failure
2. /(?v:ruby)/ =~ "uby" # success! return
3. /(?v:ruby)/ =~ "by" # success! (illustration)
4. /(?v:ruby)/ =~ "y" # success! (illustration)
5. /(?v:ruby)/ =~ "" # failure (illustration)

Of course, Oniguruma stops at the first success (step 2). I added the rest of the steps to illustrate how it continues trying to match the rest of the input when a non-anchored regexp fails.

I had encountered this problem previously when coding OP_NEGATE, and solved it by returning a special value (ONIG_MISMATCH_FROM_NEGATE). Now I simply re-used that existing logic for (!?) expanded in (?v).
I have added your examples to the test_regexp.rb suite now.

Thanks for your consideration.

#26 - 11/16/2011 02:03 PM - sunaku (Suraj Kurapati)

I am uncertain whether /(?v:ruby)/ =~ "ruby" should return nil or 1.

What do you think?

#27 - 11/16/2011 02:42 PM - sunaku (Suraj Kurapati)

Alright, I have decided that partly negated regexps should behave like Tanaka’s 2007 solution. They are easier to reason about (consistently) in that form:

"(?v:ruby)+" =~ %q(“ruby”) # yields nil
"(?v:ruby)+" =~ %q(“rubyperl”) # yields nil
"(?v:ruby)+" =~ %q(“perlruby”) # yields nil
"(?v:ruby)+" =~ %q(abc“perlruby”xyz) # yields 3 and $& is %q(“perlruby”)  

Please review my simplified patch (attached) that no longer uses OP_NEGATE.

Thanks for your consideration.

#28 - 11/17/2011 07:50 AM - sunaku (Suraj Kurapati)

Attaching patch with updated test case to illustrate how unanchored partly negated regexp matching works:

"(?v:ruby)/ =~ "ruby"  #=> 1
["r", "u", "by"] == [$`, $&, $’]  #=> true

Thanks for your consideration.

#29 - 11/17/2011 05:43 PM - naruse (Yui NARUSE)

I doubt this function under the current implementation should be flags because /ruby/v =~ "ruby"  is now useless example. I think people who want to use this negation flag should simply write (?:(?!r).).

Moreover it has a bug, for example over http://www.ruby-forum.com/topic/133413
If the suffix is not “dog” but “tv”, the regexp may be /cat((?:(?!cat).)*)tv/.
But as following, it has false negative.

```
irb(main):013:0> /cat((?:(?!cat).)*)tv/=~"cat foo bar catv"
=> nil
```

The missing piece of your proposal is a use case. 
All existing examples are too artificial. 
Design should prior to implementations, and use cases should prior to designs.

I’m interesting in the idea negation flag. 
But your proposal is limited by implementation.

Use cases I know is:

- comments of C Language: /* … */
- SGML CDATA: &lt;! [CDATA[ … ]]&gt;
- HTML 2.0 (RFC 1866) 3.2.5. Comments: /< ![CDATA[ … ]]> 
- HTML 4.0/XML Comments: /<![CDATA[ … ]]> 
- HTTP header: until CRLFCRLF 
- sequences (lines, sentences, paragraphs, and so on) which doesn’t include a word

#30 - 11/18/2011 01:33 AM - sunaku (Suraj Kurapati)

Yui NARUSE wrote:

I doubt this function under the current implementation should be flags because /ruby/v =~ "ruby"  is now useless example. I think people who want to use this negation flag should simply write (?:(?!r).).
I respectfully disagree; for me, wholly negated regexps are more useful than partly negated ones. Please see my use cases below.

Moreover it has a bug, for example over http://www.ruby-forum.com/topic/133413 If the suffix is not "dog" but "tv", the regexp may be /cat((?!cat).)*/tv/. But as following, it has false negative.

```
irb(main):013:0> /cat((?!cat,.)*)tv/=~"cat foo bar catv"
=> nil
```

It works if you add a word-boundary anchor at the end of "cat":

```
/cat((?!cat(\b).)*)tv/=~"cat foo bar catv"
0
$&
"cat foo bar catv"
```

The missing piece of your proposal is a use case. All existing examples are too artificial. Design should prior to implementations, and use cases should prior to designs.

Very true, thanks for this much needed criticism.

I'm interesting in the idea negation flag. But your proposal is limited by implementation.

I only have use cases for wholly negated regexps (/.../v):

- some_enumerable.grep(/.../v)
- some_string =~ some_regexp # where some_regexp given by user
- case some_string; when /.../v; end

That is why I became confused when implementing partly negated regexps (/(?v:)/).

Use cases I know is

- comments of C Language: /* ... */
- SGML CDATA: <![CDATA[ ... ]>]
- HTML 2.0 (RFC 1866) 3.2.5. Comments: /<![-]+-->/
- HTML 4.0/XML Comments: /<![-]+-->/
- HTTP header: until CRLF\n\nsequences (lines, sentences, paragraphs, and so on) which doesn't include a word

These seem like good use cases of partly negated regexps (/(?v:)/).

#31 - 11/26/2011 09:15 PM - naruse (Yui NARUSE)

Suraj Kurapati wrote:

Yui NARUSE wrote:

I doubt this function under the current implementation should be flags because /ruby/v =~ "ruby" is now useless example. I think people who want to use this negation flag should simply write (?:(?!\r).).

I respectfully disagree; for me, wholly negated regexps are more useful than partly negated ones. Please see my use cases below.

Ah, you think /v is still wholly negated regexp, i see.

Moreover it has a bug, for example over http://www.ruby-forum.com/topic/133413 If the suffix is not "dog" but "tv", the regexp may be /cat((?!cat,.)*)tv/. But as following, it has false negative.
It works if you add a word-boundary anchor at the end of "cat":

```
/\cat((?:(?!cat\\b).)*)tv/=~"cat foo bar catv"
```

```
0
$s&
"cat foo bar catv"
```

This `\b` hack can only work when "t" and "v" is the same kind. When replace "v" to "t", this won't work.

```
/\cat((?:(?!cat\b).)*)t!/~"cat foo bar cat!"
```

```
=> nil
```

The missing piece of your proposal is a use case. All existing examples are too artificial. Design should prior to implementations, and use cases should prior to designs.

Very true, thanks for this much needed criticism.

I'm interesting in the idea negation flag. But your proposal is limited by implementation.

I only have use cases for wholly negated regexps (/.../v):

- some_enumerable.grep(/.../v)
- some_string =~ some_regexp # where some_regexp given by user
- case some_string; when /.../v; end

That is why I became confused when implementing partly negated regexps (/(!?v)/).

They seems reasonable.
If you suggested only wholly one with such use case, this discussion would be more simple.

---

#32 - 12/03/2011 06:22 AM - sunaku (Suraj Kurapati)
Interesting. Thanks for your feedback. I will submit a new patch that only contains wholly negated regexps (/.../v) this weekend. Cheers.

#33 - 06/08/2012 09:43 PM - naruse (Yui NARUSE)
- Status changed from Assigned to Feedback

#34 - 06/10/2012 12:10 AM - nobu (Nobuyoshi Nakada)
=begin
What will [[[Regexp#match]]] with [[[!v]]] flag return, and what will set to [[[!$]]]?
=end

#35 - 10/25/2012 11:22 PM - yhara (Yutaka HARA)
- Target version changed from 2.0.0 to 2.6

#36 - 04/11/2015 05:56 AM - akr (Akira Tanaka)
- Related to Feature #11049: Enumerable#grep_v (inversed grep) added

#37 - 02/12/2017 12:40 PM - k_takata (Ken Takata)
Onigmo 6.1.1 was merged by r57603.
It supports absent operator (?~pattern) which can be a replacement of (?v:pattern).

#38 - 09/15/2017 02:11 PM - naruse (Yui NARUSE)
- Status changed from Feedback to Closed
## Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>5588_regexp_v.patch</td>
<td>15.7 KB</td>
<td>11/13/2011</td>
<td>sunaku (Suraj Kurapati)</td>
</tr>
<tr>
<td>0001-http-redmine.ruby-lang.org-issues-5588.patch</td>
<td>18.4 KB</td>
<td>11/15/2011</td>
<td>sunaku (Suraj Kurapati)</td>
</tr>
<tr>
<td>5588_negative_lookahead.patch</td>
<td>14.1 KB</td>
<td>11/16/2011</td>
<td>sunaku (Suraj Kurapati)</td>
</tr>
<tr>
<td>5588_negative_lookahead.patch</td>
<td>14.3 KB</td>
<td>11/17/2011</td>
<td>sunaku (Suraj Kurapati)</td>
</tr>
</tbody>
</table>