Possibility for using named and normal groups together in regular expressions

It should be possible to use named and normal groups together in a regular expression.

Reason: The new relative addressing possibilities for groups, \k<-n>, \k'<-n>, \g<-n>, and \g'<-n> are very helpful for writing regular subexpressions to be used via #{...} more than once in a regular expression. Example (longer explanations are only available in German on http://www.ruby-mine.de/2009/2/7/regular-ausdr%C3%BCcke-teil-7-oniguruma-und-statische-relativbez%C3%BCge):

```ruby
# encoding: Windows-1252
module Matchelements
  def bal
    return "\((?:\g<-1>)(?:\g<-2>)*?\)"
  end
end
include Matchelements
orgstrings = [
  'firstproc(x1(33, r(3, 4)), k(3, kk(3, 4)), l(3), x2(99))', # (x1, r, k, l, x2)
  'secondproc(x1(99, 5), l(77, m(44)), x2(15))',      # (x1, r, k, l, x2)
  'thirdproc(x1(66), x2(88))',             # (x1, r, k, l, x2)
  'fourthproc(x1(44), 1, 2, 3, x2(234))'   # (x1, 1, 2, 3, x2)
]
pattern = /\w+(x1(#{bal},(?>#{bal},){1,2} x2(#{bal}))\/) /
orgstrings.each do |s|
  if s.match(pattern)
    puts "      O.K.: '#{s}'"
  else
    puts "Nicht O.K.: '#{s}'"
  end
end

This works fine:

  ruby191-p0 balmusterWorks.rb
  O.K.: 'firstproc(x1(33, r(3, 4)), k(3, kk(3, 4)), l(3), x2(99))'
  O.K.: 'secondproc(x1(99, 5), l(77, m(44)), x2(15))'
  Niet O.K.: 'thirdproc(x1(66), x2(88))'
  Niet O.K.: 'fourthproc(x1(44), 1, 2, 3, x2(234))'

One Problem is still open, because in the regular expression, that uses the subexpressions, their groups still count. If one wants to extract parts of a match normal groups are necessary, which numbers must be known - e.g. /\(#\{group\}(\[0-9]+)#\{group\})/.

In this case the usage of the result of ([0-9]+) is only possible, if one knows the number of the group. This is not visible from /\(#\{group\}(\[0-9]+)#\{group\})/, because the number of groups used in #\{group\} can only be seen by looking at the definition, which can be somewhere.

A good solution is the usage of a named group /\(#\{group\}(?\[0-9]+)#\{group\})/, but then it is no longer possible to use normal groups.
together with relative access in the definition of regular subexpressions.

It would be very helpful to allow both in one regular expression.
=end

Related issues:
Related to Ruby master - Feature #1201: Add relative group reference in back ... Closed 02/25/2009
Related to Ruby 1.8 - Feature #4239: Let's begin a talk for "1.8.8" -- How's ... Closed 01/06/2011

History
#1 - 02/25/2009 01:18 AM - WoNaDo (Wolfgang Nádasi-Donner)
=begin
Typo: I've written "[0-9}+" instead of "[0-9]+" several times, sorry.
=end

#2 - 05/06/2009 09:41 AM - akr (Akira Tanaka)
=begin
I'd like to use simple paren as a shy group.
If we allow both named and unnamed capturing group in a regexp, I recommend unnamed capturing have a new syntax, such as (?<>...).
=end

#3 - 09/18/2009 04:25 AM - marcandre (Marc-Andre Lafortune)
- Assignee set to matz (Yukihiro Matsumoto)
- Target version changed from 1.9.1 to 2.0.0
=begin
=end

#4 - 09/14/2010 04:38 PM - shyouhei (Shyouhei Urabe)
- Status changed from Open to Assigned
=begin
=end

#5 - 02/14/2012 08:12 AM - naruse (Yui NARUSE)
- Status changed from Assigned to Rejected
- Assignee deleted (matz (Yukihiro Matsumoto))

Mixed regexp both named and unnamed capturing group is disallowed by design of Ruby.