Ruby master - Feature #15865

`<expr> in <pattern>` expression

05/21/2019 01:58 AM - mame (Yusuke Endoh)

Status:	Closed
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

How about adding a syntax for one-line pattern matching: <expr> in <pattern> ?

```
[1, 2, 3] in x, y, z #=> true (with assigning 1 to x, 2 to y, and 3 to z)
[1, 2, 3] in 1, 2, 4 #=> false
```

More realistic example:

```
json = {
  name: "ko1",
  age: 39,
  address: { postal: 123, city: "Taito-ku" }
}

if json in { name:, age: (20..), address: { city: "Taito-ku" } }
  p name #=> "ko1"
else
  raise "wrong format"
end
```

It is simpler and more composable than "case...in" when only one "in" clause is needed. I think that in Ruby a pattern matching would be often used for "format-checking", to check a structure of data, and this use case would usually require only one clause. This is the main rationale for the syntax I propose.

Additional two small rationales:

- It may be used as a kind of "right assignment": 1 + 1 in x behaves like x = 1 + 1. It returns true instead of 2, though.
- There are some arguments about the syntax "case...in". But if we have <expr> in <pattern>, "case...in" can be considered as a syntactic sugar that is useful for multiple-clause cases, and looks more natural to me.

There are two points I should note:

- <expr> in <pattern> is an expression like <expr> and <expr>, so we cannot write it as an argument: foo(1 in 1) causes SyntaxError. You need to write foo((1 in 1)) as like foo((1 and 1)). I think it is impossible to implement.
- Incomplete pattern matching also rewrites variables: [1, 2, 3] in x, 42, z will write 1 to the variable "x". This behavior is the same as the current "case...in".

Nobu wrote a patch: https://github.com/nobu/ruby/pull/new/feature/expr-in-pattern

Related issues:

Related to Ruby master - Feature #14912: Introduce pattern matching syntax	**Assigned**
Related to Ruby master - Feature #16182: Should `expr in a, b, c` be allowed ...	**Open**
Related to Ruby master - Feature #16355: Raise NoMatchingPatternError when `e...	**Closed**

Associated revisions

Revision 3cee9980 - 09/26/2019 06:10 AM - nobu (Nobuyoshi Nakada)

[EXPERIMENTAL] Expression with modifier in

[Feature #15865]

History

#1 - 05/21/2019 08:21 AM - shevegen (Robert A. Heiler)

I don't have a big pro/contra opinion per se on the whole topic of pattern matching,

but I would wait a bit before finalizing more and more additions on top of it. Now pattern matching is probably there to say, I understand that, but it may still be better to not add lots of further building blocks to it. Another smaller reason is that at the least I find pattern matching to be somewhat more complex to understand; I understand that the suggestion here is simpler than long case/in statements, but I am still not sure if it would be a good idea to add too much to ideas/changes in a short period of time, without seeing how it may be used in "real" production code.

#2 - 05/21/2019 04:22 PM - mame (Yusuke Endoh)

- Description updated

Oops, the first example was wrong. Fixed.

```
-[1, 2, 3] in 1, 2, 3 #=> false
+[1, 2, 3] in 1, 2, 4 #=> false
```

#3 - 05/21/2019 06:48 PM - Eregon (Benoit Daloze)

RHS assignment looks a bit weird to me.

I guess most languages have <pattern> SIGIL/KEYWORD <expr>, which seems more natural like:

```
x, y, z = [1, 2, 3]
{ name:, age: } = json

if { name:, age: (20..), address: { city: "Taito-ku" } } = json
  p name
end
```

But I guess = is not available for this or causes too many syntax conflicts?
Maybe another sigil or keyword could be used?

The current order is also opposite of for's order, which seems confusing:

```
for k, v in h
  p [k, v]
end

h.first in k, v
p [k, v]
```

However, one can argue it's consistent with the case ... in order I suppose (but it's on different lines, so there is not really a RHS assignment):

```
case h.first
in k, v
  p [k, v]
end
```

FWIW I feel this feature is somewhat similar to the Oz programming language's local [A B] = Xs in {Show A} end construct (which keeps assignments on the LHS by convention).

#4 - 05/22/2019 07:38 AM - ko1 (Koichi Sasada)

Trivial comment.

We can't introduce guard pattern (pat if expr) because it will conflict with expr if cond.

#5 - 05/22/2019 08:04 AM - nobu (Nobuyoshi Nakada)

ko1 (Koichi Sasada) wrote:

> We can't introduce guard pattern (pat if expr) because it will conflict with expr if cond.

You have to write as val in expr and cond.

#6 - 05/22/2019 08:54 AM - matz (Yukihiro Matsumoto)

I am pretty positive about adding single line pattern matching in Ruby. I am still wondering whether if is a right keyword for it. Let me think about it for a while.

Matz.

#7 - 05/22/2019 10:10 AM - Eregon (Benoit Daloze)

matz (Yukihiro Matsumoto) Do you mean in rather than if for the keyword?

#8 - 06/04/2019 09:56 PM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote:

> - Incomplete pattern matching also rewrites variables: [1, 2, 3] in x, 42, z will write 1 to the variable "x". This behavior is the same as the current "case...in".

This sounds concerning to me.
With case/in, it's clear where the variables should be defined, and it's a matter of fixing it so the variables are only defined in that in pattern branch, and nil in other branches.
(IMHO we should fix that for 2.7, otherwise it will be a compatibility issue).

But here it's unclear how long variables in inline patterns should live.
Probably for the whole method? Or just for the if?

E.g.:

```
json = { name: "Me", age: 28, hobby: :ruby }
if json in { name:, age: (...20) }
  ...
end

if json in { name:, hobby: }
  # BUG: age should not be set here
  puts "Hello #{name}, you enjoy #{hobby} and are #{age || "unknown"} years old"
end
```

When used as if expr in pattern, I think it is natural to expect no significant side effects, but changing local variables for a partial match seems kind of problematic.

#9 - 06/14/2019 11:40 PM - ktsj (Kazuki Tsujimoto)

- Related to Feature #14912: Introduce pattern matching syntax added

#10 - 06/27/2019 12:21 AM - pvande (Pieter van de Bruggen)

As suggested by Yusuke on Twitter, I'm posting a link to my own personal "wishlist" around pattern matching. I'm happy to discuss any points that might benefit from clarification.

https://gist.github.com/pvande/822a1aba02e5347c39e8e0ac859d752b

#11 - 06/27/2019 02:47 AM - mame (Yusuke Endoh)

Thank you for your comment.

This ticket discusses one-line pattern matching: <expr> in <pattern> in my proposal. This proposal is based on the case/in pattern-matching statement. I'd like to avoid to discuss a common topic between one-line and case/in in this particular ticket.

You have three points.

1. Matching is not strict.
2. The proposed is in.
3. The pattern is on the RHS.

The design choice of (1) is done in case/in statement. The one-line version and case/in should share the semantics of each pattern. It is unarguably too confusing if they have different semantics. So, I like not to discuss this issue in this ticket.

So, you have two concerns for my proposal about one-line pattern matching syntax: (2) the keyword in and (3) the word order. I understand that it is unnatural in English. (Honestly, it is mathematically/logically correct as you said, so it works for me, a non-native, though.) However, I believe that this is the best choice for some reasons:

- The appearance of case/in statement is case <expr>; in <pattern>; ... end. In a sense, the keyword in and the order that <pattern> follows the keyword in are already established, as long as we accept case/in statement.
- The order <pattern> <match-op> <expr> is very difficult (or maybe impossible?) to implement parsing rules. Perhaps we need to use <pattern-marker> <pattern> <match-op> <expr>, but it is very tough to agree with symbol or keyword of <pattern-marker> and <match-op>.

You can discuss the grand design of pattern matching and the syntax of case/in statement in #14912. If the grand design is overturned, my proposal will need to change drastically. (But in short, I respect the design decision. I'm not fully satisfied with the syntax, but we spent very long time to discuss what keyword is suitable and feasible, and finally chose the design including the keyword in.)

#12 - 06/27/2019 06:23 AM - pvande (Pieter van de Bruggen)

mame (Yusuke Endoh) wrote:

> I'd like to avoid to discuss a common topic between one-line and case/in in this particular ticket.

Agreed; the gist discusses the entirety of the feature-space, but the conversation in this ticket should be constrained to just the one-line pattern matching proposal.

> You have three points.
>
> 1. Matching is not strict.
> 2. The proposed is in.
> 3. The pattern is on the RHS.
>
> The design choice of (1) is done in case/in statement. The one-line version and case/in should share the semantics of each pattern. It is unarguably too confusing if they have different semantics. So, I like not to discuss this issue in this ticket.

Agreed. I would expect the semantics to match, either way.

> So, you have two concerns for my proposal about one-line pattern matching syntax: (2) the keyword in and (3) the word order. I understand that it is unnatural in English. (Honestly, it is mathematically/logically correct as you said, so it works for me, a non-native, though.)

The Twitter post from ko1 that I was responding to was specifically soliciting feedback about the syntax:

From https://twitter.com/_ko1/status/1135788844916195329:

> We want to ask English speaker about this proposal. Do you feel natural on this syntax?

To be clear – I do not. I'm happy to elaborate on exactly *why* not, but – as you point out – my objections are not principally to the inline syntax.

> However, I believe that this is the best choice for some reasons:
>
> - The appearance of case/in statement is case <expr>; in <pattern>; ... end. In a sense, the keyword in and the order that <pattern> follows the keyword in are already established, as long as we accept case/in statement.

I agree, and I have posted the same gist in the other ticket (as you suggested) in the hopes of sparking similar discussion there.

> - The order <pattern> <match-op> <expr> is very difficult (or maybe impossible?) to implement parsing rules. Perhaps we need to use <pattern-marker> <pattern> <match-op> <expr>, but it is very tough to agree with symbol or keyword of <pattern-marker> and <match-op>.

I can't speak directly to the parsing difficulty. From the outside, it *looks* like a combination of destructuring assignment (which deals with multiple LHS arguments), Array and Hash literals, and method arguments (e.g. required Hash keys, keyword splats). The primary difficulty, then, would seem to come from not knowing (e.g.) if the malformed Hash you just parsed was a *pattern* or a *syntax error* until you know whether or not it's followed by <match-op>.

> You can discuss the grand design of pattern matching and the syntax of case/in statement in #14912. If the grand design is overturned, my proposal will need to change drastically. (But in short, I respect the design decision. I'm not fully satisfied with the syntax, but we spent very long time to discuss what keyword is suitable and feasible, and finally chose the design including the keyword in.)

Once again, I agree. It is correct for this syntax to mirror the block form, and I have not tried to imply otherwise.

#13 - 07/11/2019 04:20 AM - osyo (manga osyo)

It's so good :)

```
users = [
    { id: 1, name: "Homu", age: 13 },
    { id: 2, name: "mami", age: 14 },
    { id: 3, name: "Mado", age: 21 },
    { id: 4, name: "saya", age: 14 },
]

users.select { @1 in { name: /m/, age: (..14) } }
# => [{:id=>1, :name=>"Homu", :age=>13}, {:id=>2, :name=>"mami", :age=>14}]

users.filter_map { name if @1 in { name:, age: (..14) } }
# => ["Homu", "mami", "saya"]
```

#14 - 07/30/2019 04:23 AM - matz (Yukihiro Matsumoto)

Alongside pattern matching in case statements, single line pattern matching must be useful too. I understand some are not fully satisfied with in operator, but it is a very tough compromise. Introducing a new keyword could break many existing programs. I don't want to do that. But I want pattern matching in the language. The operator in does not break existing programs.

Matz.

#15 - 09/26/2019 06:11 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset git|3cee99808d629c0ec493955ce8ea019d1f8a637b.

[EXPERIMENTAL] Expression with modifier in

[Feature #15865]

#16 - 09/29/2019 07:10 PM - jonathanhefner (Jonathan Hefner)

> We want to ask English speaker about this proposal. Do you feel natural on this syntax?

(I am a native English speaker.) The syntax feels confusing to me. When I read X in Y, I expect Y to be a collection of things, and X to be an element of that collection. For example, 2 in [1, 2, 3] would return true.

I understand that a pattern describes an abstract set, and a set is a collection, so I think I see the reason for choosing in. But it still feels awkward.

I also agree it is too dangerous to introduce a new keyword, unfortunately. Here are some alternatives, off the top of my head:

- <expr> >> <pattern> ("shoveling" the values into the captured variables)
- <expr> ~> <pattern>
- <pattern> <~ <expr>
- <pattern> ~= <expr> (probably too confusing with =~)
- <pattern> :~ <expr>
- <pattern> := <expr> (has a strong connotation of "assignment" in other languages)
- <pattern> :=== <expr> (as a reference to ===)
- <pattern> ?= <expr> (kind of like +=, *=, etc)
- <pattern> =? <expr> (can be negated if desired, i.e. !?)

#17 - 10/01/2019 06:25 AM - mame (Yusuke Endoh)

jonathanhefner (Jonathan Hefner) Thank you for your comment.

> - <pattern> <~ <expr>

The difficult part is that <pattern> is not distinguishable from <expr> for a parser. For example, [1, 2, x, y] is valid not only as <pattern> but also as <expr>.
So, putting a bare <pattern> before <expr> is impossible in terms of parser implementation. If we put a prefix before <pattern> (for example, in <pattern> <~ <expr>), it may be feasible.

> - <expr> >> <pattern>

>> is a valid operator in Ruby. For example, expr >> [1, 2, x, y] is ambiguous.

> - <expr> ~> <pattern>

This may be feasible, but IMO it strongly resembles right assignment rather than pattern matching.

I don't think that <expr> in <pattern> is so great, but if we have case/in, it would be the most reasonable.

#18 - 10/02/2019 09:36 PM - jonathanhefner (Jonathan Hefner)

mame (Yusuke Endoh) wrote:

> The difficult part is that <pattern> is not distinguishable from <expr> for a parser. For example, [1, 2, x, y] is valid not only as <pattern> but also as <expr>.
> So, putting a bare <pattern> before <expr> is impossible in terms of parser implementation. If we put a prefix before <pattern> (for example, in <pattern> <~ <expr>), it may be feasible.

Yes, I see! I'm sorry I missed that point from your previous comment. I do think a prefixed syntax could help it read more naturally. If the prefixed syntax was unambiguous enough, maybe = could be used as <match-op>. For example, could @[1, 2, x, y] = ary be parsed unambiguously?

> IMO it strongly resembles ... assignment rather than pattern matching

I agree, but isn't pattern matching the same as (destructuring) assignment in this context?

#19 - 11/10/2019 01:41 PM - ktsj (Kazuki Tsujimoto)

- Related to Feature #16182: Should `expr in a, b, c` be allowed or not? added

#20 - 11/20/2019 01:00 AM - ktsj (Kazuki Tsujimoto)

- Related to Feature #16355: Raise NoMatchingPatternError when `expr in pat` doesn't match added