I frequently see code that uses some value if that value satisfies a certain condition, and something else otherwise.

```
a.some_condition ? a : b
```

And in most cases, the value of a is non-nil when the condition is satisfied.

I propose to have a method, perhaps named verify, which would implemented to be equivalent to this definition:

```
class Object
def verify
  self if yield(self)
end
end
```

Then, we can write the expression above (assuming a is non-nil when the condition is satisfied) like this:

```
a.verify{|a| a.some_condition} || b
```

Perhaps it would also be useful to do something like:

```
a.verify{|a| a.some_condition}&.chaining_of_more_methods
```

The lonely guy operator staring at the dot before him is pressed hard against the wall behind him there, the "}" character!

```
leeif
```

I don't see any improvement. The new way that would be possible with the verify method is longer and more complicated than the simple and straightforward a.some_condition ? a : b.

```
a.some_condition
```

It seems useless without method chain, i.e., variable a is too simple as an example.

```

```

That is, this is a variant of yield_self.

```
foo.bar.zot.yield_self {|a| a.some_condition ? a : b}
```

```

```

But verify is long and sounds ambiguous.

```
foo.bar.zot.if?(&:some_condition) || b
```

This came to my mind, but looks magical a little.

```
foo.bar.zot.if?(&:some_condition) || b
```
I came up with an idea better than this (https://bugs.ruby-lang.org/issues/15557).
So I withdraw this proposal. Please close it.

- Status changed from Open to Closed

- Related to Feature #15557: A new class that stores a condition and the previous receiver added