Ruby master - Feature #14844

Future of RubyVM::AST?

06/12/2018 02:16 PM - rmosolgo (Robert Mosolgo)

Status:	Open	
Priority:	Normal	
Assignee:	yui-knk (Kaneko Yuichiro)	
Target version:		

Description

Hi! Thanks for all your great work on the Ruby language.

I saw the new RubyVM::AST module in 2.6.0-preview2 and I quickly went to try it out.

I'd love to have a well-documented, user-friendly way to parse and manipulate Ruby code using the Ruby standard library, so I'm pretty excited to try it out. (I've been trying to learn Ripper recently, too: https://ripper-preview.herokuapp.com/, https://rmosolgo.github.io/ripper_events/ .)

Based on my exploration, I opened a small PR on GitHub with some documentation: https://github.com/ruby/ruby/pull/1888

I'm curious though, are there future plans for this module? For example, we might:

- Add more details about each node (for example, we could expose the names of identifiers and operators through the node classes)
- Document each node type

I see there is a lot more information in the C structures that we could expose, and I'm interested to help out if it's valuable. What do you think?

Related issues:

Related to Ruby master - Feature #15752: A dedicated module for experimental ...	**Closed**

Associated revisions

Revision b4b22b92 - 12/14/2019 11:21 AM - Eregon (Benoit Daloze)

Clarify in the documentation that RubyVM::AbstractSyntaxTree is not stable API

- See [Feature #14844].

History

#1 - 06/12/2018 03:42 PM - shevegen (Robert A. Heiler)

Just two short comments from me, so that I do not expand the issue request/question too much.

(1) I would like to suggest to you to consider adding your question to the upcoming developer meeting; it may be best to have the core team and other core contributors comment on it, including matz and (based on RubyVM I think) koichi.

The next upcoming meeting agenda should be compiled here:

https://bugs.ruby-lang.org/issues/14769

But again, this is just a suggestion from me - it is your issue/question so of course you should decide. :)

(2) The other thing is, and I may be wrong, but I think there have been quite a few introspection-related extensions to ruby in general, by different ruby hackers/developers/core contributors. You mentioned RubyVM but I think there are a few more examples; Martin in regards to unicode and ... I think the regex engine (I forgot the particular issue but I remember it was somehow related to being able to tweak more within the regex engine). MJIT is also a bit related to more introspection, at the least indirectly, if we can control how much we can optimize where and how, at the least in the future (say, past ruby 3.0). matz mentioned one requirement/goal for mruby is in regards to systems that are constrained, but with MJIT we may be able to perhaps control more optimizations when it comes to systems that have

more RAM/cpu power/cores and so forth. So, I may be wrong, but I think several changes in ruby are to some extent related to more introspection. (The MJIT author blogged about this too, I think ... something about inline-C or like that? I forgot the details right now, sorry.)

Of course it may be best to have koichi comment on relevant parts of your question such as "a lot more information in the C structures that we could expose" (and whether matz is fine with this; I think matz is fine with it but it may be best to ask this in regards to the developer meeting altogether).

I personally love introspection.

PS: Actually, I just remembered one more change that is a bit related to introspection, at

https://github.com/ruby/ruby/blob/trunk/NEWS

added Binding#source_location. [Feature #14230]

While eval() could be used, I think Binding#source_location has a nicer API.

#2 - 06/30/2018 11:43 PM - ioquatix (Samuel Williams)

Rather than create a new issue, I want to comment here, as I've also been using the new RubyVM::AST to compute code coverage of templates (think ERB templates). In the past, the existing coverage.so was not capable to parse a string, so being able to compute it by hand using RubyVM::AST is a great boon.

When using RubyVM::AST::Node#type, the return value is a string. I wondered if it would make more sense to return a symbol. Rather than a string like "NODE_SCOPE", perhaps return a symbol like :scope. It would make it easier and perhaps more efficient to traverse the AST, and it seems more Ruby-like.

#3 - 07/02/2018 12:07 AM - ioquatix (Samuel Williams)

It would be nice to expose the class and method/function name if possible. In the case of code coverage, it would allow user to explicitly ignore functions, because we can traverse the AST and ignore every line of the function.

- NODE_CLASS, NODE_MODULE expose class/module name.
- NODE_DEFN expose method name.

#4 - 07/05/2018 04:13 AM - ioquatix (Samuel Williams)

Here is what I made using RubyVM::AST. It was useful. https://github.com/ioquatix/covered

In the end I made a regular expression to match node type. It was better than I expected.

#5 - 08/10/2018 09:26 AM - bozhidar (Bozhidar Batsov)

I'm really curious what's the purpose of this module and why wasn't in developed in collaboration with the maintainers of libraries like https://github.com/whitequark/parser and https://github.com/whitequark/ast, and the maintainers of prominent AST-based tools (e.g. https://github.com/rubocop-hq/rubocop)?

Seems to me really misguided to developed a module like this one without really discussing its design with any of its potential users and to quietly just ship it with Ruby 2.6.

That has been beyond frustrating for a very long time - a handful of people sit together somewhere and make all the shots, with the feedback from everyone often simply ignored. For any library to become really useful one should look at the problems people would normally be solving using it.

#6 - 08/17/2018 12:33 AM - mame (Yusuke Endoh)

bozhidar (Bozhidar Batsov) wrote:

> I'm really curious what's the purpose of this module and why wasn't in developed in collaboration with the maintainers of libraries like https://github.com/whitequark/parser and https://github.com/whitequark/ast, and the maintainers of prominent AST-based tools (e.g. https://github.com/rubocop-hq/rubocop)?

First of all, thank you for developing the parser gem and related tools including Rubocop.

As far as I understand, RubyVM module is completely different than other builtin modules. It exposes an access to Ruby internal for very limited purpose, such as debugging the internal, prototyping a new feature, implementing a MRI-bundled feature, etc. No compatibility is guaranteed at all; its API will arbitrarily change along with internal change. It is never intended for normal users to use it. Usefulness is not a priority for the modules under RubyVM. This fact can also be seen from RubyVM::InstructionSequence. (Its has a very long name, which also represents non-casual use, so it may be better to rename RubyVM::AST with AbstractSyntaxTree.)

In fact, Yuichiro Kaneko developped RubyVM::AST for testing a parser-related new feature, column number of each node, that he introduced in Ruby 2.5. It was originally a hidden external library. Some people (including me) requested to expose it as an internal-use module, and matz approved it. Then it is introduced into trunk as an experimental feature. Kaneko-san wanted RubyVM::AST for some study purpose (for example, finding all callsites of some function), and my motivation for the exposure is that it would be needed to import Ruby3's type system. (There are some proposals for type-checking Ruby. It is not decided which would be imported, or it is even uncertain if a AST module is really needed, but I just wanted to remove a blocker candidate in advance.)

RubyVM::AST does not decrease the value of the parser gem. RubyVM::AST is useful to investigate how MRI looks a Ruby program, but is not useful as a general Ruby program parser; it may optimize the AST by omitting some non-significant letters and restructuring the tree structure, so the result might not correspond to the original source code literally. I think that this property is not useful for Rubocop, for example.

In short, I think that RubyVM::AST is not what people expected. Users may use it just for research purpose, but must not use it in production.

#7 - 08/28/2018 01:00 AM - ioquatix (Samuel Williams)

I hope this is relevant.

I found an interesting article here: http://www.oilshell.org/blog/2016/12/11.html

It describes the process Python uses.

It looks like it's standardised in a way that might be a useful goal for RubyVM::AST.

#8 - 12/07/2018 11:43 AM - lucasbuchala (Lucas Buchala)

Hello. Sorry to step in with a frivolous suggestion, but I just noticed the name RubyVM::AbstractSyntaxTree in the RC1 announcement and was surprised by the length of the word when the shorter name "AST" would suffice.

mame (Yusuke Endoh) wrote:

> [...] This fact can also be seen from RubyVM::InstructionSequence. (Its has a very long name, which also represents non-casual use, so it may be better to rename RubyVM::AST with AbstractSyntaxTree.)

Following this naming practice/convention is really necessary and beneficial?

#9 - 12/20/2018 04:28 AM - ioquatix (Samuel Williams)

After playing around with the RC2 release, I think it's pretty good.

The only thing I'd like to see, is the use of Node instances even for cases where Array is currently used. Rather than Node.type, use explicit classes.

In addition to this rather than using Array for composite nodes (e.g. when parsing module), which is pretty fragile, classes specific to the node being parsed would be great, e.g. ModuleNode which has things like #name. Rather than an array which contains a symbol somewhere.

#10 - 01/26/2019 11:06 AM - ioquatix (Samuel Williams)

I started playing around with parser gem, and I actually found it really great too. I think I'll use the parser gem going forward, since it works on many versions of Ruby. That being said, exposing Ruby's AST is also really fun.

#11 - 04/07/2019 07:07 PM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote:

> As far as I understand, RubyVM module is completely different than other builtin modules.
> It exposes an access to Ruby internal for very limited purpose, such as debugging the internal, prototyping a new feature, implementing a MRI-bundled feature, etc. No compatibility is guaranteed at all; its API will arbitrarily change along with internal change. It is never intended for normal users to use it. Usefulness is not a priority for the modules under RubyVM.
>
> RubyVM::AST does not decrease the value of the parser gem. RubyVM::AST is useful to investigate how MRI looks a Ruby program, but is not useful as a general Ruby program parser; it may optimize the AST by omitting some non-significant letters and restructuring the tree structure, so the result might not correspond to the original source code literally.
>
> In short, I think that RubyVM::AST is not what people expected.
> Users may use it just for research purpose, but must not use it in production.

mame (Yusuke Endoh) Could you summarize this in the documentation of RubyVM::AbstractSyntaxTree?

Currently, it doesn't mention anything about being not stable, being not so appropriate for source code analysis, etc:

```
AbstractSyntaxTree provides methods to parse Ruby code into
abstract syntax trees. The nodes in the tree
are instances of RubyVM::AbstractSyntaxTree::Node.
```

I think that such an AST module could have the potential to be portable, but it would need to be much more fledged out.
Relying on the order of children to access AST node fields is impractical and brittle.

I think we should have nodes with methods to access each part/field, such as ModuleDefinitionNode#body, ModuleDefinitionNode#name, etc.

#12 - 04/07/2019 07:16 PM - Eregon (Benoit Daloze)

BTW, bozhidar (Bozhidar Batsov) discussed relevant points about Ripper and RubyVM::AbstractSyntaxTree in
https://metaredux.com/posts/2019/03/30/the-missing-ruby-code-formatter.html#the-impact-of-the-parser

He makes good points there.

#13 - 04/18/2019 10:25 PM - Eregon (Benoit Daloze)

Eregon (Benoit Daloze) wrote:

> I think we should have nodes with methods to access each part/field, such as ModuleDefinitionNode#body, ModuleDefinitionNode#name, etc.

A simpler alternative would be something like RubyVM::AbstractSyntaxTree::Node#[] to access fields of a node by name.
This would make the API much easier to use, using hardcoded offsets is not good for evolution and is really hard to use (even with pattern matching!).

Actually, we already have names, but they only seem to be shown with #pp:

```
[3] pry(main)> node = RubyVM::AbstractSyntaxTree.parse("def foo; 42; end")
=> (SCOPE@1:0-1:16
 tbl: []
 args: nil
 body:
   (DEFN@1:0-1:16
    mid: :foo
    body:
      (SCOPE@1:0-1:16
       tbl: []
       args:
         (ARGS@1:7-1:7
          pre_num: 0
          pre_init: nil
          opt: nil
          first_post: nil
          post_num: 0
          post_init: nil
          rest: nil
          kw: nil
          kwrest: nil
          block: nil)
       body: (LIT@1:9-1:11 42))))
[7] pry(main)> node.children[2].children[0]
=> :foo
[8] pry(main)> node[:body][:mid]
NoMethodError: undefined method '[]' for #<RubyVM::AbstractSyntaxTree::Node:SCOPE@1:0-1:16>
[9] pry(main)> node.dig(:body, :mid)
NoMethodError: undefined method 'dig' for #<RubyVM::AbstractSyntaxTree::Node:SCOPE@1:0-1:16>
```

This would make AbstractSyntaxTree easier to use, more portable and possible to evolve without breaking code.

Note that some of these names seem a bit obscure and would be worth to be renamed to be clearer, e.g., tbl which seems to be the list of local variables.
(Similrly, mid could be method_id like in TracePoint)

FWIW, it seems many people are already using AbstractSyntaxTree, and most likely are not aware of the drawbacks (not stable API, cannot evolve the API safely currently, MRI-specific currently, might not represent the source faithfully, etc). Here is just one example:
https://github.com/oracle/truffleruby/issues/1671

#14 - 05/15/2019 09:37 PM - Eregon (Benoit Daloze)

- Assignee set to yui-knk (Kaneko Yuichiro)

There was some discussion on Twitter about RubyVM::AbstractSyntaxTree, however so far no answer from its maintainers:
https://twitter.com/eregontp/status/1125314952368218112

It seems clear RubyVM::AbstractSyntaxTree is not for serious use with the current API.
I think the fact it's in core sounds like it's the new "blessed" AST for Ruby,
but it's not that, it's experimental, cannot evolve without breaking code (see above) and rather impractical currently.

What are the advantages over Ripper for instance, which is available on other Ruby implementations?
mame (Yusuke Endoh)yui-knk (Kaneko Yuichiro) In which situations should RubyVM::AST be used instead of Ripper? Can you give examples?

So, could we make it much clearer what RubyVM::AST is intended for, in the documentation?
Very few people know that long class names mean "experimental", I believe we need something more explicit in the documentation.
Also, long class names are obviously not very efficient to tell people not to use except for intended cases.

#15 - 05/17/2019 01:05 AM - mame (Yusuke Endoh)

Eregon (Benoit Daloze) wrote:

> There was some discussion on Twitter about RubyVM::AbstractSyntaxTree, however so far no answer from its maintainers:
> https://twitter.com/eregontp/status/1125314952368218112

Sorry for not replying.

> It seems clear RubyVM::AbstractSyntaxTree is not for serious use with the current API.
> I think the fact it's in core sounds like it's the new "blessed" AST for Ruby,
> but it's not that, it's experimental, cannot evolve without breaking code (see above) and rather impractical currently.

You are almost correct.
"Not for serious use" and "experimental" are a bit different than what I meant, though.

I think my opinion is clear in https://bugs.ruby-lang.org/issues/14844#note-6, but I repeat it shortly.
In my opinion, the API is mainly for research and debugging purpose. (Used on development or in ruby/test, for example.)
Also, it may be used to develop a tool that strongly depends upon a specific Ruby version. (Used in stdlib, for example.)
It is not intended for casual use.
(Note that all the above is just for my opinion.)

> What are the advantages over Ripper for instance, which is available on other Ruby implementations?
> mame (Yusuke Endoh)yui-knk (Kaneko Yuichiro) In which situations should RubyVM::AST be used instead of Ripper? Can you give examples?

Ripper does not create an AST. It is just a "tracer" of how the parser works.
There is Ripper.sexp, but it does not reproduce the details including parser-level optimization.
In theory, we may improve Ripper to be identical to the actual parser, but it is very tough.
Rather, wrapping and returning the actual AST is much easier to implement and always precise.
(Honestly, I want to remove Ripper that makes the parser code very messy. I know it is impossible, though.)

#16 - 05/17/2019 12:56 PM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) Thank you for the reply.

Could you or yui-knk (Kaneko Yuichiro) propose a description to include in the documentation, summarizing what was said?

Could you also give your opinion on accessing Node members by name (https://bugs.ruby-lang.org/issues/14844#note-13) ?

> Ripper does not reproduce the details including parser-level optimization.

What kind of details? Could you give an example?
Things like OPCALL instead of CALL? Is that useful for any tool?

I tried a simple expression to compare Ripper and RubyVM::AbstractSyntaxTree:

```
pry(main)> Ripper.sexp("def m(a) a * 2 end")

=> [:program,
 [[:def,
   [:@ident, "m", [1, 4]],
   [:paren, [:params, [[:@ident, "a", [1, 6]]], nil, nil, nil, nil, nil, nil]],
   [:bodystmt, [[:binary, [:var_ref, [:@ident, "a", [1, 9]]], :*, [:@int, "2", [1, 13]]]], nil, nil, nil]]]

pry(main)> RubyVM::AbstractSyntaxTree.parse("def m(a) a * 2 end")

=> (SCOPE@1:0-1:18
 tbl: []
 args: nil
 body:
   (DEFN@1:0-1:18
    mid: :m
    body:
      (SCOPE@1:0-1:18
```

```
       tbl: [:a]
       args:
         (ARGS@1:6-1:7
          pre_num: 1
          pre_init: nil
          opt: nil
          first_post: nil
          post_num: 0
          post_init: nil
          rest: nil
          kw: nil
          kwrest: nil
          block: nil)
       body: (OPCALL@1:9-1:14 (LVAR@1:9-1:10 :a) :* (ARRAY@1:13-1:14 (LIT@1:13-1:14 2) nil)))))
```

Indeed, the RubyVM::AbstractSyntaxTree version seems easier to read (and access once we have RubyVM::AST::Node#[:field_name]).
I think one of the main gains is node fields are named, while they are just a flat Array in Ripper.sexp.

OTOH, things are far from perfectly clear (so I think "experimental/not for serious use" seems appropriate currently).
For instance, one has to manually associate arguments given as e.g. a number for pre_num and their names in tbl.
Optional arguments seem exposed more clearly, by having node under ARGSnode[:opt], however the OPT_ARG look nested like a cons-list instead of being an Array which would be more intuitive.

So if we compare a slightly more complex example with the parser gem, we see there are lots of opportunities to make RubyVM::AST easier to access/process/read/understand:

```
pry(main)> require 'parser/current'

pry(main)> RubyVM::AbstractSyntaxTree.parse("def m(b,a,c=3,d=4) a * 2 end")

=> (SCOPE@1:0-1:28
 tbl: []
 args: nil
 body:
   (DEFN@1:0-1:28
    mid: :m
    body:
      (SCOPE@1:0-1:28
       tbl: [:b, :a, :c, :d]
       args:
         (ARGS@1:6-1:17
          pre_num: 2
          pre_init: nil
          opt: (OPT_ARG@1:10-1:17 (LASGN@1:10-1:13 :c (LIT@1:12-1:13 3)) (OPT_ARG@1:14-1:17 (LASGN@1:14-1:17
:d (LIT@1:16-1:17 4)) nil))
          first_post: nil
          post_num: 0
          post_init: nil
          rest: nil
          kw: nil
          kwrest: nil
          block: nil)
       body: (OPCALL@1:19-1:24 (LVAR@1:19-1:20 :a) :* (ARRAY@1:23-1:24 (LIT@1:23-1:24 2) nil)))))

pry(main)> Parser::CurrentRuby.parse("def m(b,a,c=3,d=4) a * 2 end")

=> s(:def, :m,
  s(:args,
    s(:arg, :b),
    s(:arg, :a),
    s(:optarg, :c,
      s(:int, 3)),
    s(:optarg, :d,
      s(:int, 4))),
  s(:send,
    s(:lvar, :a), :*,
    s(:int, 2)))
```

I think it would be good to take inspiration from parser here, which makes it really convenient to access the AST and still seems to not lose any important information.

In fact, in what cases the additional things in RubyVM::AST such as the SCOPE nodes would be useful beyond debugging the MRI parser?
Would any tool be able to do anything with those that it could not without?

I understand exposing the internal AST directly is the simplest implementation-wise.

But I think it's quite sub-optimal to access, process and understand.
Would it be better to expose an AST more similar, or even exactly the same, as the parser gem?

#17 - 05/17/2019 04:22 PM - mame (Yusuke Endoh)

Eregon (Benoit Daloze) wrote:

> Could you or yui-knk (Kaneko Yuichiro) propose a description to include in the documentation, summarizing what was said?

I think that it is stated very clearly in the doc of RubyVM module.

```
    /*
     * Document-class: RubyVM
     *
     * The RubyVM module provides some access to Ruby internals.
     * This module is for very limited purposes, such as debugging,
     * prototyping, and research.  Normal users must not use it.
     */
```

> Could you also give your opinion on accessing Node members by name (https://bugs.ruby-lang.org/issues/14844#note-13) ?

I have no strong opinion, but am a bit negative. I'm afraid if making it too useful may wrongly attract casual users.
And, if the feature requires an additional code for the node definition, it may be cumbersome to tweak the node definition.

> > Ripper does not reproduce the details including parser-level optimization.
>
> What kind of details? Could you give an example?
> Things like OPCALL instead of CALL? Is that useful for any tool?

For example, foo(**{}) is currently removed at the parser. Ripper does not.

```
$ ruby -rripper -e 'pp RubyVM::AbstractSyntaxTree.parse("foo()")'
(SCOPE@1:0-1:5 tbl: [] args: nil body: (FCALL@1:0-1:5 :foo nil))

$ ruby -rripper -e 'pp RubyVM::AbstractSyntaxTree.parse("foo(**{})")'
(SCOPE@1:0-1:9 tbl: [] args: nil body: (FCALL@1:0-1:9 :foo nil))

$ ruby -rripper -e 'pp Ripper.sexp("foo()")'
[:program,
 [[:method_add_arg, [:fcall, [:@ident, "foo", [1, 0]]], [:arg_paren, nil]]]]

$ ruby -rripper -e 'pp Ripper.sexp("foo(**{})")'
[:program,
 [[:method_add_arg,
   [:fcall, [:@ident, "foo", [1, 0]]],
   [:arg_paren,
    [:args_add_block,
     [[:bare_assoc_hash, [[:assoc_splat, [:hash, nil]]]]],
     false]]]]]
```

I don't think this is a bug of Ripper. Rather, this reflects the difference between their purposes. I think Ripper provides an AST that is closer to the original code. RubyVM::AST provides what the interpreter sees. (It is arguable whether the omission of **{} is good or bad. Anyway, it is another story.)

Another example is "for" statement. The actual parser has 20+ lines for the syntax, but Ripper code has only one line.
https://github.com/ruby/ruby/blob/ea3e7e268546599883b25d9a33d26e042461ac25/parse.y#L2796

I'm unsure if this makes an important difference for any tool, but I think it is important for a purpose to research and debug MRI.

#18 - 05/17/2019 07:53 PM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote:

> Eregon (Benoit Daloze) wrote:
>
> > Could you or yui-knk (Kaneko Yuichiro) propose a description to include in the documentation, summarizing what was said?
>
> I think that it is stated very clearly in the doc of RubyVM module.

I think that is not enough, because when people look at the documentation of RubyVM::AbstractSyntaxTree,
they don't necessarily look at the documentation of every parent module.
In fact, I find this relation rather unclear (does documentation of a parent module apply to a nested module), and for instance RubyVM::InstructionSequence is used in bootsnap
and that doesn't fit in "debugging, prototyping, and research" so much.
In my PR to clarify the purpose of RubyVM (https://github.com/ruby/ruby/pull/2113/files), I repeated everywhere that RubyVM is MRI-specific and I think we should do the same to say all of these are experimental.
Maybe we can do this with "As part of RubyVM, this module is MRI-specific and experimental".

Most importantly, I would like to see some discussion on when to use RubyVM::AbstractSyntaxTree instead of Ripper or the parser gem in the documentation.
This seems extremely important to me for people looking at RubyVM::AbstractSyntaxTree.
This is I think what we must address the most urgently for RubyVM::AbstractSyntaxTree.

> I have no strong opinion, but am a bit negative. I'm afraid if making it too useful may wrongly attract casual users.

There are already many usages of RubyVM::AbstractSyntaxTree, and yet I think almost none of them is aware of the pitfalls discussed here.
I think there is no good way to prevent users to use available features.
IMHO the only way is not having the feature, or not by default.
Maybe this is what should have happened for RubyVM::AbstractSyntaxTree: be behind a ./configure flag, disabled by default until it's ready to be used by Rubyists at large.

I think accessing by name is what would allow RubyVM::AbstractSyntaxTree to evolve in a compatible way.
Without that, I think almost every AST or parser change will break users of RubyVM::AbstractSyntaxTree.

#19 - 05/22/2019 07:41 AM - akr (Akira Tanaka)

We are not sure the stability of RubyVM::AbstractSyntaxTree.
For example, Ruby 2.7 will add new node for pattern match.

We want to know such unstability has big impact for
practical applications or not.

I feel it is difficult to decide stable definition of AST now.
I think what we can now is adding some warning in document.

#20 - 05/22/2019 10:15 AM - Eregon (Benoit Daloze)

akr (Akira Tanaka) wrote:

> We are not sure the stability of RubyVM::AbstractSyntaxTree.
> For example, Ruby 2.7 will add new node for pattern match.

Right, so I think we need to document it's not stable yet as clearly as possible.

> We want to know such unstability has big impact for
> practical applications or not.

New nodes are probably fine, but reordering node fields for instance I guess would break most usages given the current API.

> I feel it is difficult to decide stable definition of AST now.

I understand, I'm not asking a stable definition.
But I'd like to see in the documentation mentions of use-cases where using RubyVM::AbstractSyntaxTree over alternatives would make sense.
That way, I hope we can make it clear for some use-cases using RubyVM::AbstractSyntaxTree is not the right tool, or at least not always the best tool for it.

> I think what we can now is adding some warning in document.

I think that would be a good step for 2.7.

#21 - 12/14/2019 11:24 AM - Eregon (Benoit Daloze)

- Status changed from Open to Closed

Applied in changeset git|b4b22b9278007b106fe40c0191f8dcf5e7e8c0f2.

Clarify in the documentation that RubyVM::AbstractSyntaxTree is not stable API

- See [Feature #14844].

#22 - 12/14/2019 11:31 AM - Eregon (Benoit Daloze)

- Status changed from Closed to Open

The ticket was closed unintentionally, I reopen it.
What's the way to mention an issue in a commit but not close it?

I clarified in the documentation that RubyVM::AbstractSyntaxTree is not stable API:
https://github.com/ruby/ruby/commit/b4b22b9278007b106fe40c0191f8dcf5e7e8c0f2

```
$ ri RubyVM::AbstractSyntaxTree

AbstractSyntaxTree provides methods to parse Ruby code into abstract
syntax trees. The nodes in the tree are instances of
RubyVM::AbstractSyntaxTree::Node.

This class is MRI specific as it exposes implementation details of the
MRI abstract syntax tree.

This class is experimental and its API is not stable, therefore it might
change without notice. As examples, the order of children nodes is not
guaranteed, the number of children nodes might change, there is no way
to access children nodes by name, etc.

If you are looking for a stable API or an API working under multiple
Ruby implementations, consider using the parser gem or
Ripper. If you would like to make RubyVM::AbstractSyntaxTree stable,
please join the discussion at https://bugs.ruby-lang.org/issues/14844.
```

If people read that documentation, it should be very clear now and they can make an informed decision on whether using RubyVM::AbstractSyntaxTree as it is is appropriate for their use case.

I think we should make RubyVM::AbstractSyntaxTree stable longer term, but for that we need to address the mentioned issues about how to evolve the API in a compatible way when the internal AST changes.
We should also probably not have it under RubyVM (#15752).

#23 - 12/14/2019 11:50 AM - Eregon (Benoit Daloze)

BTW, I think it would be good to include other Ruby implementations (cc headius (Charles Nutter)enebo (Thomas Enebo)) in this discussion, since I think we all want to support a stable API for ASTs longer term.

As a concrete case, actionview_precompiler is using RubyVM::AbstractSyntaxTree and to be compatible with JRuby it now also has a org.jruby.ast backend:
https://github.com/jhawthorn/actionview_precompiler/commit/55007d8a09c983378c2dad88b5b3b56d608e51da

A portable stable API for ASTs would be very useful I think, and likely quite more efficient than external gems parsing Ruby code.

#24 - 08/01/2020 01:01 PM - Eregon (Benoit Daloze)

- Related to Feature #15752: A dedicated module for experimental features added

#25 - 08/01/2020 01:11 PM - Eregon (Benoit Daloze)

RBS is using RubyVM::AbstractSyntaxTree for rbs prototype.
RBS is an official project under the Ruby organization, so I believe it should be able to fully work on other Ruby implementations too (issue about this in RBS).
That is impossible as long as AbstractSyntaxTree is under RubyVM, or as long as RBS uses RubyVM::AbstractSyntaxTree with no fallback.

So, I think it is time to move AbstractSyntaxTree outside of RubyVM.
Otherwise, other Ruby implementations will have no choice but to define RubyVM too, which everyone seems to agree is unwanted as RubyVM is (at least currently) meant CRuby-only.

I see three ways forward:

- Move AbstractSyntaxTree under ExperimentalFeatures (#15752), so it is still experimental but at least other Ruby implementations can implement it too.
- Make AbstractSyntaxTree stable, and move it under a stable namespace (maybe just ::AbstractSyntaxTree?)
- Change the meaning of RubyVM so it is not CRuby-specific but also exists on other Ruby implementations. Many people don't know that RubyVM means experimental, so ExperimentalFeatures seems much clearer.

I prefer the first option, but any of the 3 unblocks the situation (which is also explained in #15752).

Can we pick one?

matz (Yukihiro Matsumoto) can you decide?

#26 - 08/08/2020 08:35 AM - ioquatix (Samuel Williams)

I was using RubyVM::AbstractSyntaxTree but moved to parser gem. I cannot see any reason to use RubyVM::AbstractSyntaxTree unless you are concerned about some specific details of the current implementation's parser.

#27 - 08/08/2020 02:25 PM - Eregon (Benoit Daloze)

ioquatix (Samuel Williams) Two reasons I've heard from projects using RubyVM::AbstractSyntaxTree are: built-in (not an extra dependency), and performance.
But "built-in (not an extra dependency)" also means only works on CRuby 2.6+, and some versions of RubyVM::AbstractSyntaxTree have bugs that might never be fixed.
And the performance of parser seems fairly reasonable.

#28 - 08/31/2020 08:22 AM - matz (Yukihiro Matsumoto)

I am OK with move the class from RubyVM::AbstractSyntaxTree. We have to decide the new name for it (RubyAST?).
In addition, the structure of the abstract syntax tree may be slightly changed from time to time (by refactoring or syntax addition).

Matz.