How to write a linter? [closed]

Asked 16 years, 3 months ago Modified 14 years, 8 months ago

Viewed 20k times

26

Closed. This question is off-topic. It is not currently accepting answers.

Want to improve this question? Update the question so it's on-topic for Stack Overflow.

Closed 12 years ago.

Improve this question

In my day job I, and others on my team write a lot of hardware models in Verilog-AMS, a language supported primarily by commercial vendors and a few opensource simulator projects. One thing that would make supporting each others code more helpful would be a LINTER that would check our code for common problems and assist with enforcing a shared code formatting style. I of course want to be able to add my own rules and, after I prove their utility to myself, promote them to the rest of the team.. I don't mind doing the work that has to be done, but of course also want to leverage the work of other existing projects.

Does having the allowed language syntax in a yacc or bison format give me a leg up? or should I just suck each language statement into a perl string, and use pattern matching to find the things I don't like?

(most syntax and compilation errors are easily caught by the commercial tools.. but we have some of our own extentions.)

coding-style verilog syntax-checking

Share

Improve this question

Follow

edited Apr 4, 2010 at 4:59

Jon Seigel

12.4k ● 8 ● 60 ● 93

asked Sep 16, 2008 at 9:26

jbdavid

622 ● 1 ● 7 ● 9

7 Answers

Sorted by: Highest score (default)

20

lex/flex and yacc/bison provide easy-to-use, well-understood lexer- and parser-generators, and I'd really recommend doing something like that as opposed to doing it procedurally in e.g. Perl. Regular expressions are powerful stuff for ripping apart strings with relatively-, but not totally-fixed structure. With any real programming language, the size of your state machine gets to be simply unmanageable with anything short of a Real

Lexer/Parser (tm). Imagine dealing with all possible interleavings of keywords, identifiers, operators, extraneous parentheses, extraneous semicolons, and comments that are allowed in something like Verilog AMS, with regular expressions and procedural code alone.

There's no denying that there's a substantial learning curve there, but writing a grammar that you can use for flex and bison, and doing something useful on the syntax tree that comes out of bison, will be a much better use of your time than writing a ton of special-case string-processing code that's more naturally dealt with using a syntax-tree in the first place. Also, what you learn writing it this way will truly broaden your skillset in ways that writing a bunch of hacky Perl code just won't, so if you have the means, I highly recommend it ;-)

Also, if you're lazy, check out the Eclipse plugins that do syntax highlighting and basic refactoring for Verilog and VHDL. They're in an incredibly primitive state, last I checked, but they may have some of the code you're looking for, or at least a baseline piece of code to look at to better inform your approach in rolling your own.

answered Sep 19, 2008 at 5:50

community wiki
Matt J

4

Don't underestimate the amount of work that goes into a linter. Parsing is the easy part because you have tools (bison, flex, ANTLR/PCCTS) to automate much of it.

But once you have a parse, then what? You must build a semantic tree for the design. Depending on how complicated your inputs are, you must elaborate the Verilog-AMS design (i.e. resolving parameters, unrolling generates, etc. If you use those features). And only then can you try to implement rules.

I'd seriously consider other possible solutions before writing a linter, unless the number of users and potential time savings thereby justify the development time.

Share Improve this answer

Follow

answered Jun 15, 2009 at 18:20

d3jones

781 ● 3 ● 5

Amen. See semanticdesigns.com/Products/DMS/DMSToolkit for generlized analysis machinery that goes beyond just parsing. DMS is used to construct custom tools. Even so, building a Linter for VerilogAMS would still be a pretty big task. – Ira Baxter Sep 5, 2009 at 3:32

2

I've written a couple verilog parsers and I would suggest PCCTS/ANTLR if your favorite programming language is C/C++/Java. There is a PCCTS/ANTLR Verilog grammar that you can start with. My favorite parser generator is Zebu which is based on Common Lisp.

Of course the big job is to specify all the linting rules. It makes sense to make some kind of language to specify the linting rules as well.

Share Improve this answer

Follow

answered Nov 16, 2008 at 21:23

user37248

257 ● 2 ● 4

0

In trying to find my answer, I found this on ANTLR - might be of use

Share Improve this answer

Follow

answered Sep 16, 2008 at 9:29

Chris Kimpton

5,541 ● 6 ● 48 ● 73

0

If you use Java at all (and thus IDEA), the IDE's extensions for custom languages might be of use

Share Improve this answer

Follow

answered Sep 16, 2008 at 9:30

Chris Kimpton

5,541 ● 6 ● 48 ● 73

My "IDE" is Cadence Library Manager + Nedit, with the (builtin) Verilog syntax highlighting, adding my own patterns to support Verilog-A and Verilog-AMS. I don't see much value in Java for the kind of work I'm doing. – jbdavid Sep 16, 2008 at 9:36

0

`yacc` / `bison` definitely gives you a leg up, since good linting would require parsing the program. Regex (true regex, at least) might cover trivial cases, but it is easy to write code that the regexes don't match but are still bad style.

Share Improve this answer

Follow

answered Sep 16, 2008 at 9:30

Hank Gay

71.8k • 36 • 161 • 222

0

ANTLR looks to be an alternative path to the more common (OK *I* heard about them before) YACC/BISON approach, which it turns out also commonly use LEX/FLEX as a front end.

a Quick read of the FLEX man page kind of make me think It could be the framework for that regex type of idea..

Ok.. I'll let this stew a little longer, then see how quickly I can build a prototype parser in one or the other.

and a little bit longer

Share Improve this answer Follow

edited Oct 3, 2008 at 18:45

answered Sep 16, 2008 at 10:14

jbdavid

622 ● 1 ● 7 ● 9