Macho II: Programming With Man Pages

Anthony Cozzie, Sam King at UIUC

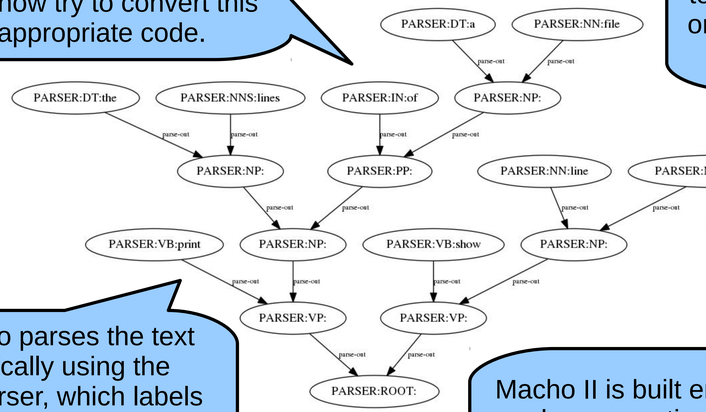
Problem:

Because programming is hard, we have a lot of tools to help automate the process, like static bug checking (Coverity), databases (Prospector,Sniff), or even automatic debuggers. But most developers don't use them: it's a huge effort to keep up with a rapidly changing field now, because these tools cannot be purely automatic: they need the developer to verify decisions, like 'does this API really do what I want?' These are fundamental issues until we develop mind reading machines, so most programmers only use Emacs, Google, and their brains.

Solution:

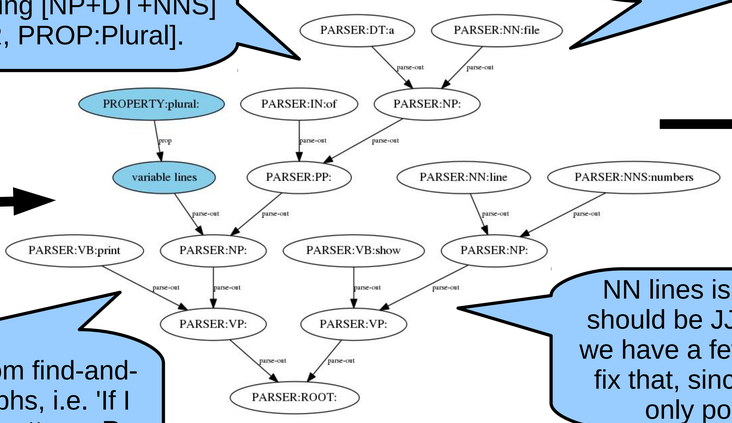
Why not just run all the tools together and verify the (many) suggested programs against an example? Getting even one example correct provides an end-to-end check that all the tools worked with high probability. Since many of the decisions are dependent (like selecting multiple library calls) we can reduce the programmer's workload, as well as provide a simple, general interface to which new tools can be added transparently. Macho starts from natural language and tries to automate as much of the programming process as possible. Of course it is still fairly limited – sadly Star Trek isn't here yet.

Our example is cat: “Print the lines of a file. Show line numbers.” Macho will now try to convert this text to appropriate code.



First, Macho parses the text grammatically using the Stanford Parser, which labels each word as a noun (NN), plural noun (NNS), verb phrase (VP), and so on.

Macho converts plural nouns to variables (probably arrays or lists) using [NP+DT+NNS] → [VAR, PROP:Plural].

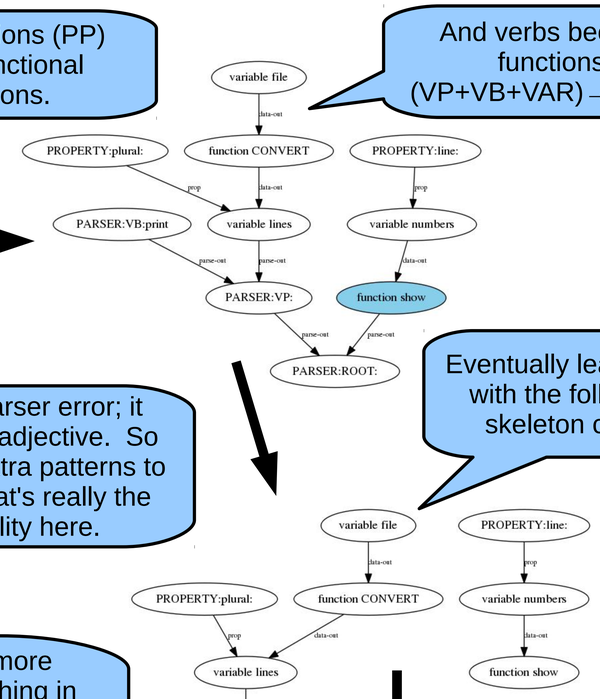


Macho II is built entirely from find-and-replace operations on graphs, i.e. 'If I have nodes **X** that match patterns **P**, replace them with arbitrary new nodes **Y**. Changes are highlighted in blue.

And prepositions (PP) become functional conversions.

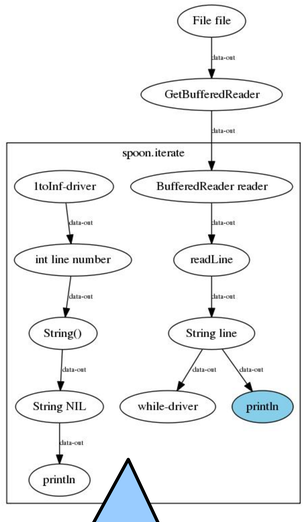
NN lines is a parser error; it should be JJ for adjective. So we have a few extra patterns to fix that, since that's really the only possibility here.

And verbs become functions (VP+VB+VAR) → FUNC

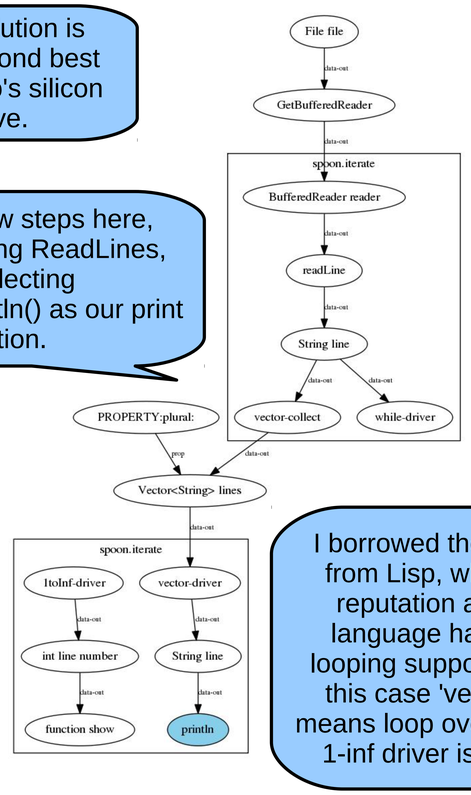


Eventually leaving us with the following skeleton code.

The inlined solution is actually the second best one from Macho's silicon perspective.

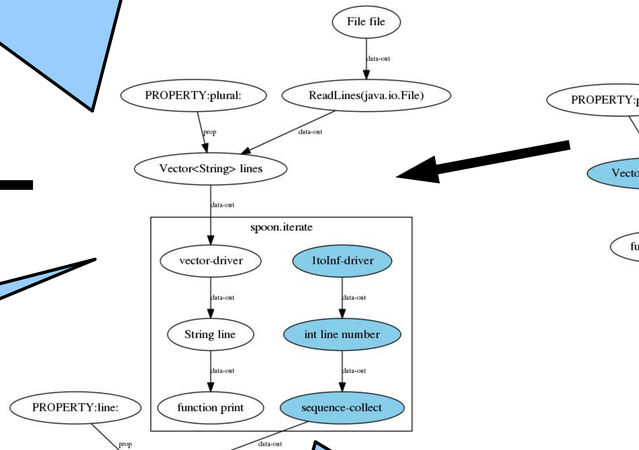


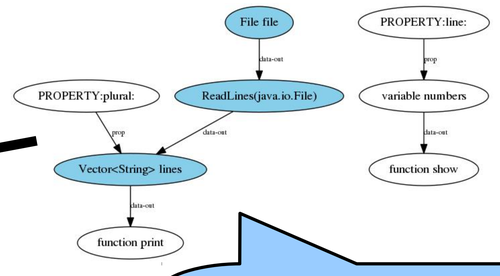
Skipping a few steps here, including inlining ReadLines, and selecting System.out.println() as our print function.



Macho combines the two loops here even though they both have side effects! Hooray for end-to-end checks on correctness!

The graph/pattern system gives Macho II a lot more flexibility than Macho I, which tried to mine everything in an unsupervised way out of open source Java code. It's fairly easy to define tricky patterns like 'line numbers' manually (in this case 'line numbers' + 'line variable in loop') that would be extremely hard to learn automatically.





Now Macho uses its database of code to fill in unlabeled functions based on types and variable names (just another pattern).

I borrowed the 'iterate' macro from Lisp, which despite its reputation as a functional language has much better looping support than Java. In this case 'vector-driver' just means loop over the vector, and 1-inf driver is just a counter.

The various solutions are ranked by a Bayesian model learned from the collection of code, though.

I'm looking for a job! If you are interested in paying me a lot of money to work on Macho things, say Hi!