

Prolog Coding Guidelines: Status and Tool Support

Falco Nogatz, Philipp Körner, [Sebastian Krings](#)

University of Würzburg, University of Düsseldorf, Niederrhein University of Applied Sciences

Coding Guidelines

- Importance of coding guidelines generally accepted
- Establishing fraught with obstacles:
 - ▶ Finding common ground on kind and selection of rules
 - ▶ Adhering to or enforcing rules is complicated
 - ▶ Not all rules deemed sensible can be automatically verified

State of Prolog

Work towards unified coding guidelines for Prolog by

- Covington et al. (2012),
- McLaughlin (1990),
- Kaplan (1991).

But how widespread are those rules in the community?

State of Prolog

Work towards unified coding guidelines for Prolog in

- Coding guidelines for Prolog,
- A Plea for a Readable Prolog Programming Style,
- A Plea for Readable Pleas for a Readable Prolog Programming Style.

Last title especially stresses what this is all about!

Key Point

- Our key point: widespread acceptance only with automation
- Covington agrees: “2.17 Consider using automated tools to improve readability”

Approach

- Check to what extend rules can be automated
- Implement linter including those rules
- Use linter to verify state of the community

Layout Rules

- Mostly easy to verify automatically
- However, some rules rely on personal opinion or subjective assessment
 - ▶ Use layout to make comments more readable
 - ▶ Avoid comments to the right of a line of code, unless inseparable from the Lines

Naming Conventions

Similar problems occur when trying to verify naming conventions:

- Can of course check underscore-style / camel case / etc.
- But:
 - ▶ Rules regarding the pronunciation of identifiers?
 - ▶ Rules taking semantics into account, e.g., use a special name for predicates representing relations
 - ▶ Add the types on which a predicate operates to its name

Language idioms

- Deciding steadfastness is impossible in general
 - ▶ Common pattern: unification of a parameter after a cut
- Without knowing type or mode of inputs cannot decide on order of arguments
- Deciding whether a cut is red or green
- Avoiding append and assert as much as possible
- Can detect body recursion but should it be replaced by tail recursion in general?
- Usage of custom implementations rather than built-in algorithms not easily detectable
- Augmenting Prolog with a run-time type and mode checking system is an idea we absolutely share. Several have been suggested.

Documentation & Methodology

Can only verify existence, except for

- Constructs that are almost always wrong
- Magic constants

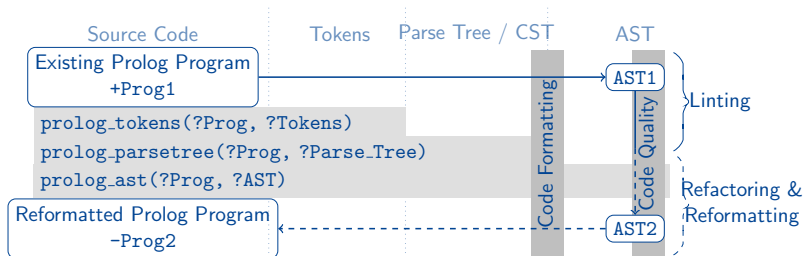
First Conclusion

- Some can be verified, some cannot
- Up next: automate!

Approaches

- Term expansion
- Existing parsers
- Existing linters
- Novel implementation

Tool Overview



Detection of SWI Extensions

- *dicts*, e.g., “point{ a: 1 }”
- *compounds without arguments*, e.g., “a()”.
- *relaxed parenthesis*, i.e., SWI Prolog allows for terms that would require parenthesis according to the ISO Prolog standard, e.g., “X = -”, and “[a :- b]” instead of “[(a :- b)]”.
- *integer exponent notation*, i.e., the ISO Prolog standard requires one to write “1.0e3” instead of “1e3”.

Own Dogfood

Prolog is nice for linting Prolog:

- EBNF rules resembles DCGs. Grammars can be directly adopted from ISO Prolog standard.
- In cases where the environment is underdetermined, e.g., if operator definitions are missing, backtracking can be used. This way, “*a b.*” can be parsed, deducing that *a/1* has to be defined as prefix operator, or *b/1* as postfix operator.
- Logic variables as arguments are bound to the correct value, e.g., a variable is bound to the maximum value of all observed line lengths.
- Predicates expresses relation between source code and corresponding tokens, parse tree and AST. Can be used in both directions without modification, i.e., to reformat.

Community Status Report

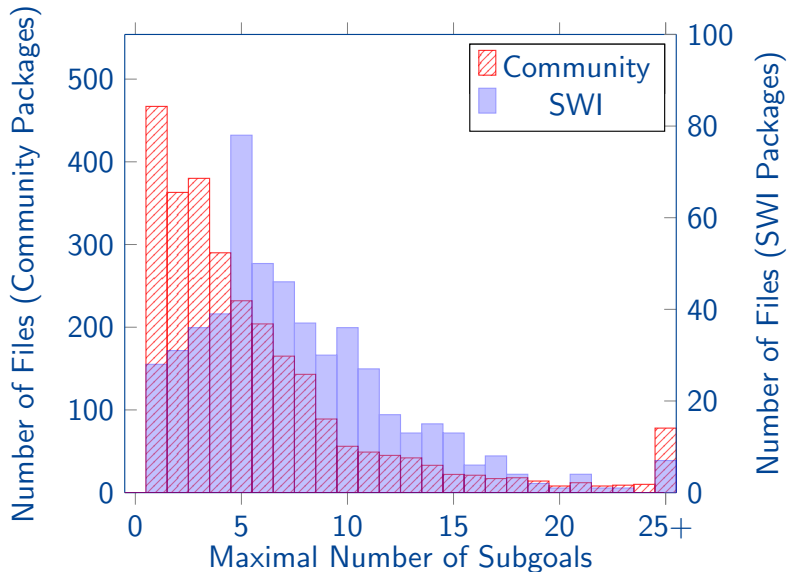
Possible scenarios:

- ① Most of the community adheres to one batch of guidelines
- ② Community is fractured, adhering to different guidelines
 - ▶ SWI vs. SICStus?
 - ▶ Certain projects, e.g., packages shipped with SWI?
- ③ Programmers have a consistent coding style
- ④ Programmers do not care and mix different coding styles

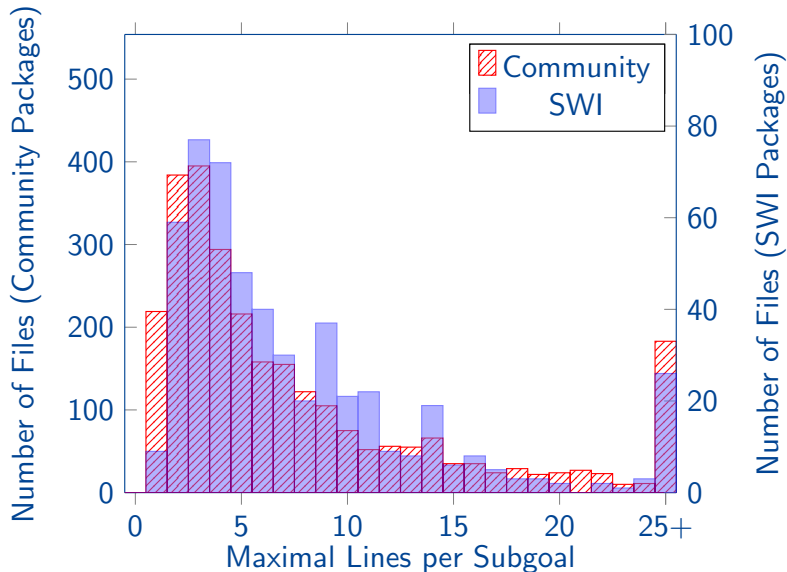
We Examine

- SWI Community Packages
 - ▶ 3815 Prolog files in 251 packages
 - ▶ 2775 files contain Rules
 - ▶ More than 90 % of the files could be parsed by our tool
- SWI Packages
 - ▶ 688 Prolog files in 34 packages
 - ▶ 530 files contain rules
 - ▶ 609 of which could be processed

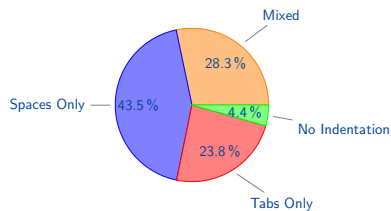
Maximal Number of Subgoals



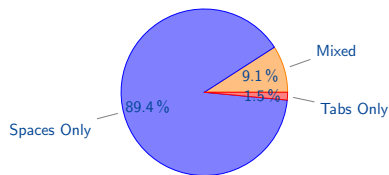
Maximal Lines per Subgoal



Indentation



(a) Community



(b) SWI

Adaptation of SWI features

Feature	Example	Community		SWI	
		Occurr.	in _ of 251 packs	Occurr.	in _ of 34 packs
shebang	<code>#!/swipl</code>	11	6 (2.4 %)	2	2 (5.9 %)
digit groups	<code>1_000</code>	77	11 (4.4 %)	4	2 (5.9 %)
dicts	<code>a{b:1}</code>	883	35 (13.9 %)	119	10 (24.4 %)
unicode character escape	<code>\u2C6F</code>	13	2 (0.8 %)	50	7 (20.6 %)
missing closing <code>"\</code>	<code>\x00</code>	9	3 (1.2 %)	0	0 (0.0 %)
single quote char constant	<code>o''</code>	13	5 (2.0 %)	1	1 (2.9 %)
zero arguments compound	<code>pi()</code>	60	8 (3.2 %)	0	0 (0.0 %)
tab in quotes	<code>"\t"</code>	159	9 (3.6 %)	0	0 (0.0 %)
integer exp. notation	<code>1e3</code>	38	8 (3.2 %)	16	3 (8.8 %)

Future Work

- Broaden scope of language features linter can be applied to
 - ▶ Add CHR, CLP(FD)
 - ▶ Need to consider, how linter interacts with term expansion
 - ▶ Need to extend guidelines
- Extend the number of rules we can check automatically
 - ▶ Language processing technology?
 - ▶ Argumentation analysis?
 - ▶ So far, we do not know about other linters implementing such techniques.
- Improve developer tools, e.g., support for LSP, usage in CI

Conclusions

- Presented a linter for Prolog supporting the coding guidelines by Covington et al. (2012) as far as we deemed feasible
- Certain rules could easily be checked automatically, other remain out of reach (for now?)
- Tool is published as SWI package and on GitHub at <https://github.com/fnogatz/plammar> (MIT License).
- Community somewhat indecisive
- New language features slow to become widespread

Thank you!

Any questions?