Developing Prolog Coding Standards

Michael A. Covington
Institute for Artificial Intelligence

Based on a paper…

Coding Guidelines for Prolog

Michael A. Covington
Roberto Bagnara
Richard A. O'Keefe
Jan Wielemaker
Simon Price

forthcoming in
Theory and Practice of Logic Programming

Based on a paper…

5 authors

>5 opinions!

My goal is to
get you thinking,
not dictate practices to you.

Developing Prolog coding standards

- Motivation
- Layout
- Naming conventions
- Documentation
- Peculiarities of Prolog
- Debugging and testing
- Conclusions

Developing Prolog coding standards

- **Motivation**
- Layout
- Naming conventions
- Documentation
- Peculiarities of Prolog
- Debugging and testing
- Conclusions

Motivation

Everybody knows this now,
but it was once news:

Programming languages are for human beings, not for computers.

The computer would be content with
0100011110001110001111100101010011...

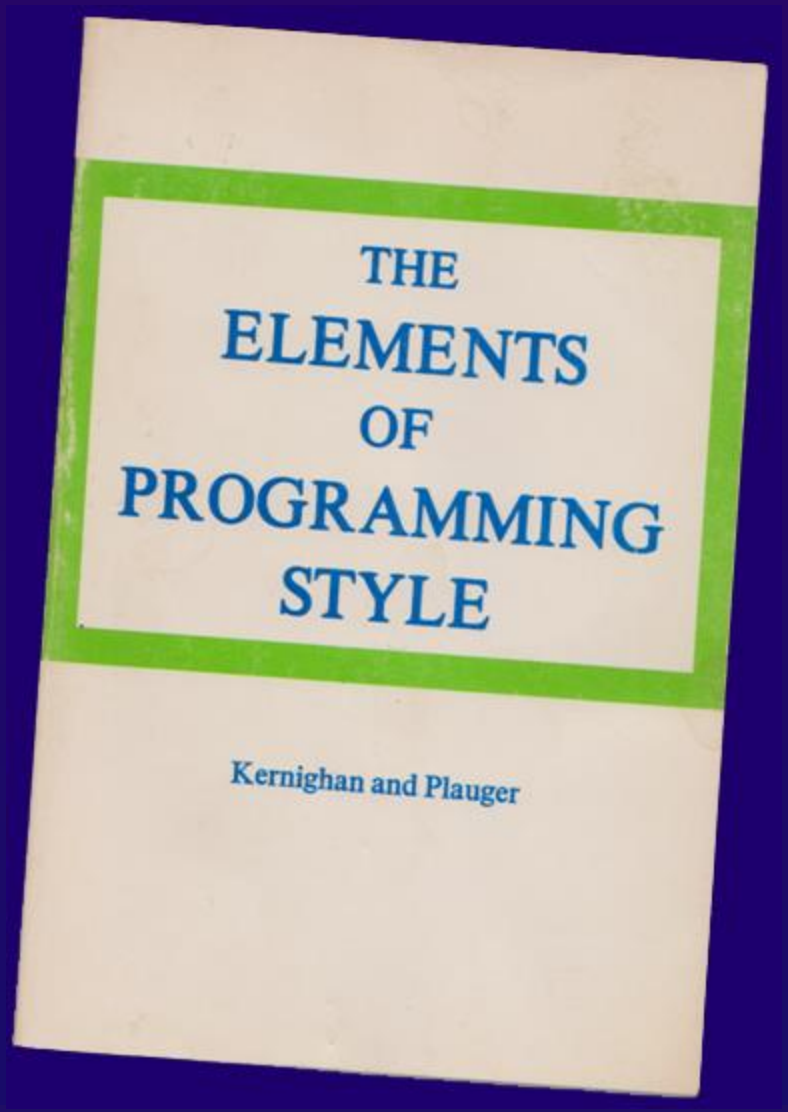
Motivation

Good coding standards

- **Make the program easier to read** (and debug)
- **Eliminate meaningless variation** (so that all changes in layout are significant)
- **Make different programmers' contributions look alike** (for coherence of a large project)

Motivation

1974...



Motivation

Kernighan and Plauger's key insight:

Writing programs for a human audience
not only
makes them easier to read,
it makes them more reliable
and even makes them run faster!

(Style ➔ thinking about what you're writing.)

Motivation

Your audience might even be
yourself
2 minutes or 2 decades later.

Motivation

So how do we apply all of this to Prolog?

Developing Prolog coding standards

- Motivation
- **Layout**
- Naming conventions
- Documentation
- Peculiarities of Prolog
- Debugging and testing
- Conclusions

Layout

The most important thing about
program layout
is to care about it
and find some way
to be neat and consistent.

Layout

Semantic principle well known to the printing industry:

Readers will expect every change in layout to mean something.

If it doesn't, you're asking them to process noise.

Layout

```
foo :- blah,
          blither,
          blither2,
       blatt.
```

Nobody can look at this without wondering why 2 subgoals are indented further.
If there's no reason, you've annoyed them.

Layout

Indent consistently, preferably by 4 spaces.

Put each subgoal on a separate line.

```
ord_union_all(N, Sets0, Union, Sets) :-
    A is N / 2,
    Z is N - A,
    ord_union_all(A, Sets0, X, Sets1),
    ord_union_all(Z, Sets1, Y, Sets),
    ord_union(X, Y, Union).
```

Layout

Consider using a proportional-pitch font.

More readable, and less like an 80-column punched card!

```
ord_union_all(N, Sets0, Union, Sets) :-
    A is N / 2,
    Z is N – A,
    ord_union_all(A, Sets0, X, Sets1),
    ord_union_all(Z, Sets1, Y, Sets),
    ord_union(X, Y, Union).
```

Layout

Make your editor store the indentations as spaces, not tabs,
so they won't change when the file is opened with another editor.

Layout

Limit the lengths of lines
(the printing industry says 55 characters;
computer industry standard of 80
is definitely too big).

Layout

Limit the number of lines per predicate (break up complex ones as needed).

Use more vertical space between different predicates than between clauses of the same predicate.

Layout

Arrange comments for readability.

Don’t write lists as continuous prose.

```
% This predicate classifies C as whitespace (ASCII < 33),
% alphabetic (a-z, A-Z), numeric (0-9), or symbolic (all
% other characters).

% This predicate classifies C as:
% - whitespace (ASCII < 33);
% - alphabetic (a-z, A-Z);
% - numeric (0-9); or
% - symbolic (all other characters).
```

Layout

Space after commas that separate goals or arguments, but not list elements.

```
pred([A,B], [D,E]) :-
  goal(A, D),
  goal(B, E).
```

The comma has 3 uses in Prolog,
and any help disambiguating it is welcome.

Layout

Make semicolons and if-then-elses very prominent through indentation.

```
pred(A) :-
   (test1(A) ->
       goal1(A)
       ;
       goal2(A)
   ).
```

One of several ways to do it

Most of us have been trained to overlook semicolons!

Layout

Consider using a prettyprinter for hardcopy.

Consider implementing a good prettyprinter for us, if you're so inclined!

Or at least fix up my old PLTeX...

Prettyprinter: LaTeX "listings" package

```
%% sum_list(+Number_List, ?Result)
%  Unifies Result with the sum the numbers in Number_List;
%  calls error/1 if Number_List is not a list of numbers.

sum_list(Number_List, Result) :-
    sum_list(Number_List, 0, Result).

%  sum_list(+Number_List, +Accumulator, ?Result)

sum_list([], A, A).          % At end: unify with accumulator.
sum_list([H|T], A, R) :-     % Accumulate first and recur.
  number(H),
  !,
  B is A + H,
  sum_list(Rest, B, R).
sum_list(_, _A, _R) :-       % Catch ill-formed arguments.
  error('first arg to sum_list/2 not a list of numbers').
```

Prettyprinter: Covington's PLTeX

```
%% sum_list(+Numbers_List, ?Result)
%   Unifies Result with the sum the numbers in Numbers_List;
%   calls error/1 if Numbers_List is not a list of numbers.

sum_list(Numbers_List, Result) ←
      sum_list(Numbers_List, 0, Result).

% sum_list(+Numbers_List, +Accumulator, ?Result)

sum_list([], A, A).                    % At end: unify with accumulator.
sum_list([H|T], A, R) ←                % Accumulate first and recur.
   number(H),
   !,
   B is A + H,
   sum_list(Rest, B, R).
sum_list(_, _A, _R) ←                  % Catch ill-formed arguments.
   error('first␣arg␣to␣sum_list/2␣not␣a␣list␣of␣numbers').
```

Developing Prolog coding standards

- Motivation
- Layout
- **Naming conventions**
- Documentation
- Peculiarities of Prolog
- Debugging and testing
- Conclusions

Naming conventions

Don’t worry, the rest of the sections
of this talk
are going to be a bit shorter!

Naming conventions

Capitalization practices are imposed on us by Prolog syntax.

Variables begin with capital letters.

Atoms and predicate names begin with lowercase letters.

(I know there are ways to get around this…)

Naming conventions

Use underscores to separate words in names.

```
Like_This

like_this
```

Naming conventions

Make names meaningful.

To convey "sort the list and count its elements"
name your predicate
`sort_and_count`, **not** `stlacie`.

To avoid typing long names,
use a short name temporarily,
then search-and-replace.

Naming conventions

Make all names pronounceable.

`xkcd` is a good name for a comic strip –
not for a Prolog predicate.

Don't use multiple names
likely to be pronounced alike.

People remember pronunciations, not spellings.

Worst I've ever seen: `MENU2`, `MENUTWO`, `MENUTOO`.

Naming conventions

Within names, don't express numbers as words.

`pred1, pred2` *not* `pred_one, pred_two`

This is a tactic to make spellings predictable from pronunciations.

Naming conventions

Don’t use digits for words.

`list_to_tree  not  list2tree`

unless you’re going for a high-school L33TSP33K look!

Naming conventions

Mark auxiliary predicates with suffixed
`_aux, _loop, _case, _1, _2`
or the like.

```
foo(…) :- …, foo_aux(…), …

foo_aux(…) :- …
```

Naming conventions

If a predicate tests a property or relation, give it a name that is *not* a command to do something.

```
sorted_list, parent
well_formed, ascending
between_limits
contains_duplicates, has_sublists
```

Naming conventions

If a predicate is best understood as an action, give it a name that *is* a command to do something.

```
remove_duplicates
print_contents
sort_and_classify
```

Naming conventions

Choose predicate names to help show the argument order.

`parent_child(X,Y)`

rather than

`parent(X,Y)`

(which is the parent of which?)

Naming conventions

Use descriptive variable names

`Input_List, Tree, Result, Count`

Decide how to use single-character names

For very localized use:

- `C` could be a single character
- `I, J, K, M, N` could be integers
- `L` could be a list

Naming conventions

Establish practices for naming threaded state variables (intermediate results).

```
foo(State0, State) :-
        foo_step(State0, State1),
        foo(State1, State).
```

No digit after the *final* name.

Naming conventions

Singular and plural are often handy for the first and rest of a list.

`[Tree|Trees]` **or even** `[T|Trees]`

Avoid
`[First|Rest], [Head|Tail], [H|T]`
unless you really can't say what is in the list.

Developing Prolog coding standards

- Motivation
- Layout
- Naming conventions
- **Documentation**
- Peculiarities of Prolog
- Debugging and testing
- Conclusions

Documentation

Begin every predicate (except auxiliary predicates) with a descriptive comment.

```
%% remove_duplicates(+List, -Processed_List) is det
%
% Removes the duplicates in List, giving Processed_List.
% Elements are considered to match if they can
% be unified with each other; thus, a partly uninstantiated
% element may become further instantiated during testing.
% If several elements match, the last of them is preserved.
```

If you can't do this, you're not ready to code the predicate.

Documentation

Use meaningful names and mode specifiers on arguments in the comment.

```
%% remove_duplicates(+List, -Processed_List)
```

Simplest system:

- `+` expected to be instantiated upon entry
- `-` expected not to be instantiated upon entry
- `?` may or may not be instantiated upon entry

Documentation

You can also specify determinism...

```
%% remove_duplicates(+List, -Processed_List) is det
```

det	Succeeds exactly once and leaves no backtrack points.
semidet	Either fails or is det.
nondet	May either fail or succeed any number of times, may leave backtrack points.
multi	Like nondet but must succeed at least once.

Documentation

Auxiliary predicates do not need full descriptive comments.

That's why they're auxiliary –
they're not called from elsewhere.

But always leave enough comments
to make it clear what you're doing.

Developing Prolog coding standards

- Motivation
- Layout
- Naming conventions
- Documentation
- **Peculiarities of Prolog**
- Debugging and testing
- Conclusions

Peculiarities of Prolog

Predicates should be steadfast.

That is, behavior should not change if the output argument is already instantiated to the right value.

```
?- foo(X), X = something.
?- foo(something).
```

Peculiarities of Prolog

Put arguments in this order:
inputs, intermediate values, outputs.

Confusing in C:

```
fprintf(stream, format, args)
fgets(string, number, stream)
```

Let's not repeat that folly.

Peculiarities of Prolog

Put the most discriminating argument first.

That is, put first the one that is most likely already to be instantiated to a value that enables you to choose clauses.

That's first-argument indexing!

Peculiarities of Prolog

Never add a cut to correct an unknown problem.

Instead, make your program logic correct without the cut, and *then* trim away unneeded searches.

Beginners often misunderstand the cut.

Peculiarities of Prolog

Work at the beginning of the list.

You can get to the first element immediately; to get to the last element you must traverse all the others.

Sometimes this means it's better to build a list backward.

Peculiarities of Prolog

Remember this idiom:

```
process_list([Old|Olds], [New|News]) :-
  do_something_to_element(Old, New),
  process_list(Olds, News).

process_list([], []).
```

This traverses the lists forward and is tail recursive. It is efficient.

Peculiarities of Prolog

Avoid `append`, `assert`, `retract`
when speed is important.

These are slower than most Prolog actions.
You may use them in prototyping
and then modify the algorithm
to speed it up when needed.

Peculiarities of Prolog

Prefer tail recursion, but don't be fanatical.

A recursive call is tail recursive if
it is the last step of a predicate and
there are no backtrack points left behind.
This saves memory – it is compiled as iteration.

Think about this if a predicate must recurse
1000 times – don't worry if it's just 10.

Developing Prolog coding standards

- Motivation
- Layout
- Naming conventions
- Documentation
- Peculiarities of Prolog
- **Debugging and testing**
- Conclusions

Debugging and testing

The best code is the one that
does the right computation,
not the one with the most tricks.

Don’t be clever.
Read the source code of UNIX and you will marvel at how un-clever it really is.

Debugging and testing

When efficiency is critical, make tests.

Perform “microbenchmarks”
to find out which way of doing something
is actually faster
with your compiler.

Use built-in predicate: `statistics.`

Debugging and testing

Look out for constructs that are nearly always wrong.

- **! at end of last clause of a predicate**
- `repeat` **not followed by !**
- `append` **with a one-element list as its first argument**

Debugging and testing

Use the built-in debugger
(`spy`, `trace`).

It's in all the textbooks,
and it still works.

Debugging and testing

Use `print` (not `write`) to add tracing output to your program.

`print` is more user-configurable than `write` (e.g., can abbreviate long lists, or handle other data specially, or even keep quiet entirely under specified conditions).

Debugging and testing

Test that every loop or repetition:

- **Starts correctly,**
- **Advances correctly from one step to the next,**
- **Ends correctly.**

This applies to all programming languages.

Debugging and testing

Test every predicate by failing back into it.

Does it do something unexpected when you make it backtrack?

Don't just test: `?- my_pred(my_arg).`
Test also: `?- my_pred(my_arg), fail.`

Getting one solution is not enough!

Debugging and testing

In any error situation, either
correct the problem or
make the program crash (not just fail).

Prolog's "failure" means "The answer is no."

It does not mean "I can't do the computation."

Debugging and testing

Make error messages meaningful.

“Cannot sort a(b,c) because it is not a list”

not

“Illegal argument, wrong type”

Debugging and testing

Don’t waste time checking for errors that would be caught anyway, such as arithmetic on non-numbers.

Prolog is an untyped language.
Don’t make the CPU spend all its time checking types on the remote chance that they might be wrong.

Developing Prolog coding standards

- Motivation
- Layout
- Naming conventions
- Documentation
- Peculiarities of Prolog
- Debugging and testing
- **Conclusions**

Conclusions

If you're ready to disagree with a good bit of what I said, I've done my job.

I've started you thinking.

Conclusions

There is no “one true style” for Prolog.

Coding standards are something
you (singular or plural)
need to develop on your own
and even change when the need arises.

Conclusions

But it is much better to have consistent habits that you can change than to have no consistent habits!

Developing Prolog coding standards