1 Formatting and Beautifying (ISO) Prolog

1.1 Test environment

The test environment is used as starting point to specify the different possible formattings for Prolog code. Based on this decisions, the test cases were build to guarantee the correctness of the formatter and as consequence the correctness of the produced code. Of course, the code has to be semantically and syntactic correct. Otherwise a formatter would be useless. Next, different scenarios are illustrated on the left-hand side and their corresponding formatted code on the right-hand side. On every example code of formation on the right-hand side, every formation in use is used.

Starting with an empty set of code, the formatter naturally should not add anything and left the blank line.

Figure 1: Formatting empty lines

For better readability there should be some white spaces around an operator, as you can see around + and = in the following figure 2.

```
X is 3+A.                    X is 3 + A.

X=d(2,3).                    X = d(2,3).
```

Figure 2: Formatting white spaces around operators

But if a comma is used for a set of arguments, different variations of white spaces could be confusing. We reduce needed white spaces to a minimum and clear all useless ones, as illustrated in 3. Of course an already optimal code should not become worse.

```
A = d(a ,b).                 A = d(a,b).
A = d(a, b).                 A = d(a,b).
A = d(a , b).                A = d(a,b).
A = d(a,b).                  A = d(a,b).
```

Figure 3: Formatting whitespaces in complex terms

Lists are more a complex topic and is handled different among programmers. Figure 4 shows a set of formations for lists. They always result in a style like [a,b|rest]. If the rest is empty, we can drop it.

```
[].
[a].
[a,[]].
[a|[]].
[a,b].
[a|b].
[a,[b,c]].
[a|[b,c]].
[a,b|c].
[a,b,c].
[[a,b],c].
[[a,b]|c].
one_to_3(L) :- L = .(1,.(2,.(3,[]))).
one_to_2s(L) :- L = .(1,.(2,_)).
```

```
[].
[a].
[a].
[a].
[a,b].
[a|b].
[a,[b,c]].
[a,b,c].
[a,b|c].
[a,b,c].
[[a,b],c].
[[a,b]|c].

one_to_3(L) :- L = [1,2,3].

one_to_2s(L) :- L = [1,2|_].
```

Figure 4: Formatting lists

Also white spaces in lists can occur but of course reduce readability as well. Figure 5 shows the corresponig scenario and the formatted solution.

```
[ ].
[ a].
[a ].
[ a ].
[a ,b].
[a| b].
[a, [b, c]].
[a,b ,c].
[a,b| c].
[a , b, c].
[ [a,b],c].
[[a,b] | c].
```

```
[].
[a].
[a].
[a].
[a,b].
[a|b].
[a,[b,c]].
[a,b,c].
[a,b|c].
[a,b,c].
[[a,b],c].
[[a,b]|c].
```

Figure 5: Formatting white spaces in lists

If the comma is used as an & operator, it has to be formatted in another way to see the difference to a simple enumeration of parameters. Due to this fact, commas as functors have a white space behind them as illustrated in figure 6.

```
A = a , b.
```

```
A = a, b.
```

Figure 6: Formatting comma as a functor

To be able to distinguish the different clauses and their possible different arity figure 7 shows that there should not be any empty line between same clauses with the same arity.

```
bit_xor(0,0,0).

bit_xor(0,1,1).

bit_xor(1,0,1).


bit_xor(1,1,0).
```

```
bit_xor(0,0,0).
bit_xor(0,1,1).
bit_xor(1,0,1).
bit_xor(1,1,0).
```

Figure 7: Formatting empty lines between same clause with same arity

If the arity is different but the clauses is again the same, we introduce only one empty line between them as mentioned in figure 8.

```
attack(a,b,c) :- a,b.
attack(a,b,c) :- a,b,c.
attack(a,b,c,d) :- !.
```

```
attack(a,b,c) :- a, b.
attack(a,b,c) :- a, b, c.

attack(a,b,c,d) :- !.
```

Figure 8: Formatting empty lines between same clauses with different arity

Finally, if there are to different clauses, we separate them with two empty lines as you can see in figure 9.

```
qsort([], R, R).
partition([],_,[],[]).
```

```
qsort([],R,R).


partition([],_,[],[]).
```

Figure 9: Formatting empty lines between different clauses

A very important topic are brackets in Prolog to see which operator is at first and has a higher priority. Based on simple mathematics, the next scenario handles this problem. Naturally, multiplication has a higher priority than addition in mathematics, if the addition is not in brackets. On the other hand useless brackets can be dropped. Both test cases are illustrated in figure 10.

```
   a + b.
(a + b).
a + b * c.
a + (b * c).

(a + b) * c.
(a + b) * (a + b).

a:-b,c ; d,(X is 3 * 3),!.
```

```
   a + b.
a + b.
a + b * c.
a + b * c.

(a + b) * c.
(a + b) * (a + b).

a :- b, c
;
d, X is 3 * 3, !.
```

Figure 10: Formatting needed brackets around operators

```
   (a = 1 ; b = 2).
(a;b;c).
(a;b),(c;d).
```

```
   a = 1 ; b = 2.
a ; b ; c.

(a ; b), (c ; d).
```

Figure 11: Formatting or statements

The last topic is line break. When a line reaches the default or user-set linewidth the clause has to be split to several lines. Three cases are possible. First, the line fits into the line lenght. Second the line willbe split after the :- and it will be checked if clause body and head are both fit in their lines. Last the clause will be split after the :- and at every AND (,) in the body 12.

```
  test(a,b,c).
test(a,b,c) :- a = b.
test(a,b,c) :- a = b, b = c, c = a, c is a.
test(a,b,c) :- a = b, b = c, c = a, a is b, b is c, c is a, a is 'very_complicated'.
test(a,b,c) :- a = b, b = c, c = a; a is b; b is c, c is a, a is 'very_complicated'.

  test(a,b,c).
test(a,b,c) :- a = b.
test(a,b,c) :-
  a = b, b = c, c = a, c is a.
test(a,b,c) :-
  a = b,
  b = c,
  c = a,
  a is b,
  b is c,
  c is a,
  a is very_complicated.
test(a,b,c) :-
  a = b,
  b = c,
  c = a ; a is b ; b is c,
  c is a,
  a is very_complicated.
```

Figure 12: Formatting needed brackets around operators