

# Mini-project I

Jonas Smedegaard

2026-04-08

## Problem 1

### Exercise a

That a propositional formula is satisfiable means it has at least one solution, i.e. at least one combination of involved truth variables leads to a true statement.

That a propositional formula is a tautology means it is always true, i.e. all combinations of involved truth variables lead to a true statement.

That a propositional formula is a contradiction means it is never true, i.e. no combinations of involved truth variables lead to a true statement.

### Exercise b

#### 1.2.11 b

$p \rightarrow (p \vee q)$  is a tautology iff its negation is not satisfiable.

The proposition is therefore negated, and then – since only tableau rules for negation and conjunction are permitted here – rewritten using Material nonimplication and De Morgan's 2nd law, in that order:

$$\begin{aligned} & \neg(p \rightarrow (p \vee q)) \\ & \quad \Downarrow \\ & p \wedge \neg(p \vee q) \\ & \quad \Downarrow \\ & p \wedge \neg p \wedge \neg q \end{aligned}$$

This negated and rewritten proposition is then tested for satisfiability using tableau method:

1.	$p \wedge \neg p \wedge \neg q \vee$	
2.	$p \wedge \neg p \vee$	$1 \wedge$
3.	$\neg q$	$1 \wedge$
4.	$p$	$2 \wedge$
5.	$\neg p$	$2 \wedge$
	$\otimes$	



The above tableau is complete, and its only branch is closed as it contains both formulas  $p$  and  $\neg p$ . This proves that the proposition negated is a contradiction, hence the proposition itself is a tautology.

~~1.2.11c~~

$\neg p \rightarrow (p \rightarrow q)$  is a tautology iff its negation is not satisfiable.

The proposition is therefore negated, and then – since only tableau rules for negation and conjunction are permitted here – rewritten using Material implication (twice):

$$\begin{aligned} &\neg(\neg p \rightarrow (p \rightarrow q)) \\ &\quad \Leftrightarrow \\ &\neg p \wedge \neg(p \rightarrow q) \\ &\quad \Leftrightarrow \\ &\neg p \wedge p \wedge \neg q \end{aligned}$$

This negated and rewritten proposition is then tested for satisfiability using tableau method:

1.	$\neg p \wedge p \wedge \neg q \vee$	
2.	$\neg p$	$1 \wedge$
3.	$p \wedge \neg q \vee$	$1 \wedge$
4.	$p$	$3 \wedge$
5.	$q$	$3 \wedge$
	$\otimes$	



Being logically equivalent to previous 1.2.11 b, the above tableau is complete, and its only branch is closed as it contains both formulas  $p$  and  $\neg p$ . This proves that the proposition negated is a contradiction, hence the proposition itself is a tautology.

### Exercise c

$((P \rightarrow Q) \rightarrow P) \rightarrow P$  (Peirce's law) is a tautology iff its negation has no solution:

The proposition is therefore negated, and then – since only tableau rules for negation and conjunction are permitted here – rewritten using Material implication (thrice):

$$\begin{aligned} & \neg(((P \rightarrow Q) \rightarrow P) \rightarrow P) \\ & \quad \Downarrow \\ & \neg((\neg(P \wedge Q) \rightarrow P) \rightarrow P) \\ & \quad \Downarrow \\ & \neg(\neg(\neg(P \wedge Q) \wedge P) \rightarrow P) \\ & \quad \Downarrow \\ & \neg\neg(\neg(\neg(P \wedge Q) \wedge P) \wedge P) \end{aligned}$$

This negated and rewritten proposition is then tested for satisfiability using tableau method:


1.	$\neg\neg(\neg(\neg(P \wedge Q) \wedge P) \wedge P) \vee$	
2.	$\neg(\neg(P \wedge Q) \wedge P) \wedge P \vee$	1 $\neg\neg$
3.	$\neg(\neg(P \wedge Q) \wedge P) \vee$	2 $\wedge$
4.	$P$	
5.	$\neg\neg(P \wedge Q) \vee$	3 $\neg\wedge$
	$\swarrow \quad \searrow$ $P \wedge Q \vee \quad P$	5 $\neg\neg$
6.	$P$	
7.	$Q$	
8.	$\otimes$	




The above tableau is complete with one branch open, proving that the proposition negated is not a contradiction, hence the proposition itself is not a tautology.

### Exercise d

Yes.

I cannot prove that completing a tableau is always possible for proposition logic, but cannot intuitively imagine a counter-example either. I trust Graham Priest, when in the presented chapter about 'review[ing] classical propositional logic' he writes that '[i]n the present case, the branches of a completed tableau are always finite', where I assume that by 'the present case' he is referring to propositional logic. 

This assumption, that propositional logic but not higher-order logic can be deterministically resolved in finite computing time, aligns with feedback I recieved from professors at the beginning of my bachelor when I excitedly shared that my field of interest was Semantic Web: They tried (as I recall) to explain to me, that the scientific interest in Semantic Web has faded in recent times due to higher order logic having been proven not reliably computable. 

## Problem 2

### Exercise a

For the tableau to be satisfiable, either of its two branches must be satisfiable. Left branch requires  $q$  to be both true and false and right branch that  $p$  is both true and false, neither of which is possible to satisfy, hence the tableau as a whole is unsatisfiable.




## Problem 3


### Exercise a

Given that an inhabitant of the country speaks either always true or always false, each person represents a logical truth value.

Let  $P(x)$  be 'x is a politician' and  $S(x, y)$  be 'x says y', where  $P(x) \equiv \neg x$  and  $S(x, y) \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$ .

The conversation can now be expressed in propositional logic as  $S(b, S(a, \neg P(a))) \wedge S(c, P(a))$ . 

The above ignores temporal aspects of the conversation, notably the explicitly mentioned fact that  $c$  spoke after  $b$ , because  $b$  spoke about a fact unknown to  $c$ , so knowledge of  $b$  speaking truth or not cannot affect the later statement.

More generally it might be part of the riddle whether both strangers and inhabitants 'has to take a lot of care when interacting', or if inhabitants can tell each other apart. Again this has no consequence for the concrete conversation, however, since the two statements are independent of each other. 

## Exercise b

$S(b, S(a, \neg P(a))) \wedge S(c, P(a))$  is first rewritten using  $P(x)$  equivalence (twice), double negation elimination, and  $S(x, y)$  equivalence (thrice):

$$\begin{aligned} & S(b, S(a, \neg P(a))) \wedge S(c, P(a)) \\ & \quad \Downarrow \\ & S(b, S(a, \neg\neg a)) \wedge S(c, P(a)) \\ & \quad \Downarrow \\ & S(b, S(a, \neg\neg a)) \wedge S(c, \neg a) \\ & \quad \Downarrow \\ & S(b, S(a, a)) \wedge S(c, \neg a) \\ & \quad \Downarrow \\ & S(b, (a \wedge a) \vee (\neg a \wedge \neg a)) \wedge S(c, \neg a) \\ & \quad \Downarrow \\ & ((b \wedge ((a \wedge a) \vee (\neg a \wedge \neg a))) \vee (\neg b \wedge \neg((a \wedge a) \vee (\neg a \wedge \neg a)))) \wedge S(c, \neg a) \\ & \quad \Downarrow \\ & ((b \wedge ((a \wedge a) \vee (\neg a \wedge \neg a))) \vee (\neg b \wedge \neg((a \wedge a) \vee (\neg a \wedge \neg a)))) \wedge (c \wedge \neg a) \vee (\neg c \wedge \neg\neg a) \end{aligned}$$

The proposition is tested for satisfiability using tableau method, involving also the two disjunction rules in Priest's section 1.4.4 (in addition to the three rules listed in this mini project):

1.	$((b \wedge ((a \wedge a) \vee (\neg a \wedge \neg a))) \vee (\neg b \wedge \neg((a \wedge a) \vee (\neg a \wedge \neg a)))) \wedge (c \wedge \neg a) \vee (\neg c \wedge a) \vee$				
2.	$(b \wedge ((a \wedge a) \vee (\neg a \wedge \neg a))) \vee (\neg b \wedge \neg((a \wedge a) \vee (\neg a \wedge \neg a))) \vee$	1 $\wedge$			
3.	$(c \wedge \neg a) \vee (\neg c \wedge a) \vee$	1 $\wedge$			
$\swarrow \qquad \qquad \qquad \searrow$					
4.	$b \wedge ((a \wedge a) \vee (\neg a \wedge \neg a)) \vee$ $\neg b \wedge \neg((a \wedge a) \vee (\neg a \wedge \neg a)) \vee$	2 $\vee$			
5.	$b$	$\neg b$	4 $\wedge$		
6.	$(a \wedge a) \vee (\neg a \wedge \neg a) \vee$ $\neg((a \wedge a) \vee (\neg a \wedge \neg a)) \vee$	4 $\wedge$			
$\swarrow \qquad \qquad \qquad \searrow \qquad \qquad \qquad \swarrow \qquad \qquad \qquad \searrow$					
7.	$c \wedge \neg a \vee$ $\neg c \wedge a \vee$ $c \wedge \neg a \vee$ $\neg c \wedge a \vee$	3 $\vee$			
8.	$c$	$\neg c$	$c$	$\neg c$	7 $\wedge$
9.	$\neg a$	$a$	$\neg a$	$a$	7 $\wedge$
$\swarrow \qquad \qquad \qquad \searrow \qquad \qquad \qquad \swarrow \qquad \qquad \qquad \searrow$					
10.	$a \wedge a \vee$ $\neg a \wedge \neg a$ $a \wedge a \vee$ $\neg a \wedge \neg a$	6 $\vee$			
11.	$a$	$\neg a$	$a$	$\neg a$	10 $\wedge$
12.	$a$	$\neg a$	$a$	$\neg a$	10 $\wedge$
13.	$\otimes$	$\otimes$	$\neg(a \wedge a) \vee$	$\neg(a \wedge a) \vee$	6 $\neg \vee$
14.	12,11,9	12,11,9	$\neg(\neg a \wedge \neg a)$	$\neg(\neg a \wedge \neg a) \vee$	6 $\neg \vee$
15.			$\neg \neg a$	$\neg \neg a \vee$	14 $\neg \wedge$
16.			$\neg \neg a$	$\neg \neg a \vee$	14 $\neg \wedge$
17.			$a$	$a$	16, 15 $\neg \neg$
			$\otimes$		
			14,9		

The above tableau is complete with 3 of its 6 branches open, proving that the proposition is satisfiable. The open branches show the possible solutions to the riddle:  $\{\neg a, b, c\}$ ,  $\{a, b, \neg c\}$  and  $\{a, \neg b, \neg c\}$ , i.e. Albert is either a politician or a citizen, Billy is a citizen when Albert is a citizen, and Camilla is the opposite of Albert.


(Billy being conditional to Albert is surprising, and may indicate an error somewhere in the tableau: it was expected that Albert was simply either/or)

**Exercise c**

Not understood.

# Mini-project II

Jonas Smedegaard <sup>1</sup>


<sup>1</sup>Roskilde University, Department of People and Technology 

2026-04-24

## Question 1

The truth table for the negation of Exercise 1.2.14 from Rosen's book, i.e. the formula  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$ , is shown below.

$p$	$q$	$\neg p$	$p \rightarrow q$	$\neg p \wedge (p \rightarrow q)$	$\neg q$	$(\neg p \wedge (p \rightarrow q)) \rightarrow \neg q$	$\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$
T	T	F	T	F	F	T	F
T	F	F	F	F	T	T	F
F	T	T	T	T	F	F	T
F	F	T	T	T	T	T	F

The table shows that  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  is satisfiable. Specifically, The two leftmost columns and the rightmost column show that the formula is satisfied when  $p$  is true and  $q$  is false. 

This can be expressed as the function  $v$  with domain  $\{p, q, \neg, \wedge, \rightarrow\}$ , codomain  $\{0, 1\}$  and the following mapping:

$v(\neg x) = 1$  if  $v(x) = 0$ , and 0 otherwise.

$v(x \wedge y) = 1$  if  $v(x) = 1$  and  $v(y) = 1$ , and 0 otherwise.

$v(x \rightarrow y) = 0$  if  $v(x) = 1$  and  $v(y) = 0$ , and 1 otherwise.

## Question 2



The function  $v(x \vee y)$  can be expressed in set-theoretic notation as

$$\{B_1, \dots, B \cup \{\phi \vee \psi\}, \dots, B_i\} \rightsquigarrow \{\{B_1, \dots, B \cup \{\phi\}, \dots, B_i\}, \{B_1, \dots, B \cup \{\psi\}, \dots, B_i\}\}$$

and the function  $v(x \rightarrow y)$  can be expressed as

$$\{B_1, \dots, B \cup \{\phi \rightarrow \psi\}, \dots, B_i\} \rightsquigarrow \{\{B_1, \dots, B \cup \{\neg\phi\}, \dots, B_i\}, \{B_1, \dots, B \cup \{\psi\}, \dots, B_i\}\}$$



## Question 3

TODO

## Question 4


TODO

## Question 5

TODO

# Mini-project III

Jonas Smedegaard <sup>1</sup>

<sup>1</sup>Roskilde University, Department of People and Technology 

2026-05-10

## Question 3B-1

The set of propositional formulas, expressed using the Type 2 grammar Extended Backus-Naur form (EBNF):

```
1 formula      = "(" formula ")" | compound | atomic;
2 compound    = nonequivalence | nonimplication
3              | nonconjunction | nondisjunction | double negation
4              | equivalence | implication | conjunction | disjunction;
5 nonequivalence = NOT "(" equivalence ")";
6 nonimplication = NOT "(" implication ")";
7 nonconjunction = NOT "(" conjunction ")";
8 nondisjunction = NOT "(" disjunction ")";
9 double negation = NOT NOT formula;
10 equivalence   = formula EQUIV formula;
11 implication   = formula IMPLIES formula;
12 conjunction   = formula AND formula;
13 disjunction   = formula OR formula;
14 EQUIV         = "↔";
15 IMPLIES       = "→";
16 AND           = "∧";
17 OR            = "∨";
18 NOT           = "¬";
19 atomic        = PROPOSITIONAL | NOT PROPOSITIONAL;
20 PROPOSITIONAL = "p" | "q" | "r";
```

Support for optional space before and after symbols is omitted since it is semantically irrelevant and would reduce readability.

To show that the formula in Exercise 1.2.11c fits the grammar, top-down parsing is used until terminals are reached:

1.  $\neg p \rightarrow (p \rightarrow q)$  (compound, line #1)
2.  $\neg p \rightarrow (p \rightarrow q)$  (implication, lines #2, #4)
3.  $\neg p \rightarrow (p \rightarrow q)$  (implication, line #11)
4.  $\neg p \rightarrow (p \rightarrow q)$  (atomic, line #1)
5.  $\neg p \rightarrow (p \rightarrow q)$  (IMPLIES, line #15)
6.  $\neg p \rightarrow (p \rightarrow q)$  (parenthesized formula, line #1)
7.  $\neg p \rightarrow (p \rightarrow q)$  (atomic, line #19)
8.  $\neg p \rightarrow (p \rightarrow q)$  (NOT, line #18)
9.  $\neg p \rightarrow (p \rightarrow q)$  (PROPOSITIONAL, line #20)
10.  $\neg p \rightarrow (p \rightarrow q)$  (compound, line #1)
11.  $\neg p \rightarrow (p \rightarrow q)$  (implication, lines #2, #4)
12.  $\neg p \rightarrow (p \rightarrow q)$  (implication, line #11)
13.  $\neg p \rightarrow (p \rightarrow q)$  (atomic, line #1)
14.  $\neg p \rightarrow (p \rightarrow q)$  (IMPLIES, line #15)
15.  $\neg p \rightarrow (p \rightarrow q)$  (atomic, line #1)
16.  $\neg p \rightarrow (p \rightarrow q)$  (atomic, line #19)
17.  $\neg p \rightarrow (p \rightarrow q)$  (PROPOSITIONAL, line #20)
18.  $\neg p \rightarrow (p \rightarrow q)$  (atomic, line #19)
19.  $\neg p \rightarrow (p \rightarrow q)$  (PROPOSITIONAL, line #20)

Fitting of Exercise 1.2.14 is similarly shown as top-down parsing until terminals:

1.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (compound, line #1)
2.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (nonimplication, line #2)
3.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (nonimplication, line #6)
4.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (NOT, line #18)
5.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (parenthesized formula, line #1)
6.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (IMPLIES, line #15)
7.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #1)
8.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (compound, line #1)
9.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (conjunction, lines #2, #4)
10.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (conjunction, line #12)
11.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #1)
12.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (AND, line #16)
13.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (parenthesized formula, line #1)
14.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #19)

15.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (NOT, line #18)
16.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (PROPOSITIONAL, line #20)
17.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (compound, line #1)
18.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (implication, lines #2, #4)
19.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (implication, line #11)
20.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #1)
21.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (IMPLIES, line #15)
22.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #1)
23.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #19)
24.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (PROPOSITIONAL, line #20)
25.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #19)
26.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (PROPOSITIONAL, line #20)
27.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (atomic, line #19)
28.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (NOT, line #18)
29.  $\neg((\neg p \wedge (p \rightarrow q)) \rightarrow \neg q)$  (PROPOSITIONAL, line #20)

### Question 3B-2

EBNF-derived Type 2 grammar covering an infinite set of propositional formulas:

```

1 formula      = "(" formula ")" | compound | atomic;
2 compound    = nonequivalence | nonimplication
3              | nonconjunction | nondisjunction | double negation
4              | equivalence | implication | conjunction | disjunction;
5 nonequivalence = NOT "(" equivalence ")";
6 nonimplication = NOT "(" implication ")";
7 nonconjunction = NOT "(" conjunction ")";
8 nondisjunction = NOT "(" disjunction ")";
9 double negation = NOT NOT formula;
10 equivalence   = formula EQUIV formula;
11 implication   = formula IMPLIES formula;
12 conjunction   = formula AND formula;
13 disjunction   = formula OR formula;
14 EQUIV         = "↔";
15 IMPLIES       = "→";
16 AND           = "∧";
17 OR            = "∨";
18 NOT           = "¬";

```

```

19 atomic          = propositional | NOT propositional;
20 propositional  = LETTER { LETTER } { NUMBER };
21 LETTER         = ? the regular expression [[:alpha:]] ?; (* unicode? *)
22 NUMBER        = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9";

```

The first 18 lines are identical with previous grammar (and this very limited change makes me wonder if perhaps I have misunderstood one of the questions).

### Question 3B-3

Prolog program implementing the decision procedure in Question 3 of Mini-project 2, for propositional formulas built using the connectives  $\neg$  and  $\wedge$ :

```

1  % -*- Mode: Prolog -*-
2  % SPDX-FileCopyrightText: 2026 Jonas Smedegaard <dr@jones.dk
3  % SPDX-License-Identifier: GPL-3.0-or-later
4  %
5  % Tableau-rules expansion of a propositional formula.
6  %
7  %% Usage, with REPL or non-interactively:
8  %% swipl -q formulator.prolog
9  %% swipl -g 'QUERY.' -t halt. formulator.prolog
10 %% gprolog --consult-file formulator.prolog
11 %% gprolog --query-goal "consult('formulator.prolog'), QUERY, halt"
12 %%
13 %% QUERY examples with expected response (indented):
14 %% complete([not(not(q))])
15 %%   Complete tableau: [[q]]
16 %% complete([not(and(p,and(not(not(q)),q)))]])
17 %%   Complete tableau: [[not(p),not(q),not(q)]]
18
19 % log messages to console
20 info(String) :- format('~w~n', [String]).
21 info(String, Term) :- format('~w: ~w~n', [String, Term]).
22 warn(String) :- format('Warning: ~w~n', [String]).
23 warn(String, Term) :- format('Warning: ~w: ~w~n', [String, Term]).
24

```

```

25 %% first_item(Item, List)
26 %
27 % Item is the first item of List if List is non-empty
28 first_item([Item|_], Item).
29
30 %% distribute(Formula, TableauIn, TableauOut)
31 %
32 % TableauOut is TableauIn with Formula prepended to each branch
33 distribute(_, [], []).
34 distribute(Formula, [Rest|TableauIn], [[Formula|Rest]|TableauOut]) :-
35     distribute(Formula, TableauIn, TableauOut).
36
37 %% well_formed_formula(Formula)
38 %
39 % True when Formula is a valid propositional formula
40 well_formed_formula(Formula) :-
41     atom(Formula).
42 well_formed_formula(Formula) :-
43     compound_formula(Formula).
44
45 %% well_formed(FormulaBranch)
46 %
47 % True when FormulaBranch is a list of well-formed formulas
48 well_formed([]).
49 well_formed([Formula|Rest]) :-
50     well_formed_formula(Formula),
51     well_formed(Rest).
52
53 %% compound_formula(Formula)
54 %
55 % True when Formula is a compound propositional formula
56 compound_formula(not(Formula)) :-
57     well_formed_formula(Formula).
58 compound_formula(and(Formula1, Formula2)) :-
59     well_formed_formula(Formula1),
60     well_formed_formula(Formula2).
61
62 %% expand_branch(Formula, Branch, BranchExpanded)
63 %
64 % BranchExpanded is Branch with Formula expanded and prepended

```

```

65 expand_branch(Formula, Branch, BranchExpanded) :-
66     expand([Branch], Branches),           % reuses expand/2 for branches
67     distribute(Formula, Branches, BranchExpanded).
68
69 %% has_expansion_rule(Formula)
70 %
71 % True when Formula is handled by a specific expansion rule
72 has_expansion_rule(not(not(_))).
73 has_expansion_rule(and(_,_)).
74 has_expansion_rule(not(and(_,_))).
75
76 %% expand(TableauIn, TableauOut)
77 %
78 % TableauOut is TableauIn with branches expanded
79 expand([], []).
80 expand([[_|TableauIn], [_|TableauOut]]) :-
81     expand(TableauIn, TableauOut).
82
83 % TableauOut is TableauIn with first formula of first branch expanded
84 expand([[Formula|Branch]|Rest], TableauOut) :-
85     \+ has_expansion_rule(Formula), % omit separately handled expansions
86     expand_branch(Formula, Branch, Expanded),
87     expand(Rest, ExpandedRest),
88     append(Expanded, ExpandedRest, TableauOut).
89
90 % Expansion by double negation
91 expand([[not(not(Formula))|Branch]|Rest], TableauOut) :-
92     expand([[Formula|Branch]|Rest], TableauOut).
93
94 % Expansion by conjunction
95 expand([[and(Formula1, Formula2)|Branch]|Rest], Tableau) :-
96     expand([[Formula1, Formula2|Branch]|Rest], Tableau).
97
98 % Expansion by nonconjunction
99 expand([[not(and(Formula1, Formula2))|Branch]|Rest], Tableau) :-
100     expand([[not(Formula1), not(Formula2)|Branch]|Rest], Tableau).
101
102 complete(FormulaBranch) :-
103     well_formed(FormulaBranch),
104     expand([FormulaBranch], Tableau),

```

```

105     info('Complete tableau', Tableau).
106
107 % FIXME: needs to drop contradicting branches before checking if empty
108 open(FormulaBranch) :-
109     expand(FormulaBranch, Tableau),
110     first_item(_, Tableau),
111     info('Tableau is open');
112     warn('Tableau is closed'), false.
113
114 % TODO
115 %sound(FormulaBranch) :-
116 %     well_formed(FormulaBranch),
117 %     expand([FormulaBranch], Tableau),
118 %     member(OpenBranch, Tableau),
119 %     info('Formula is sound (i.e. satisfiable)');
120 %     warn('Formula is unsound (i.e. not satisfiable)'), false.
121
122 %% testsuite for SWI Prolog
123 %
124 % Usage: swipl -g run_tests. -t halt. formulator.prolog
125 :- if(current_prolog_flag(dialect, swi)).
126 :- begin_tests(lists).
127 test(well_formed_atomic, [nondet]) :-
128     well_formed_formula(q).
129 test(unwell_formed_nothing, [nondet]) :-
130     \+ well_formed_formula([]).
131 test(unwell_formed_connective, [nondet]) :-
132     \+ well_formed_formula(non(q)).
133 test(well_formed_compound, [nondet]) :-
134     well_formed_formula(not(not(q))).
135 test(unwell_formed_compound, [nondet]) :-
136     \+ well_formed_formula(and(not(q))).
137 test(not, [nondet]) :-
138     expand([[not(not(q))]], [[q]]).
139 test(not_not, [nondet]) :-
140     expand(
141         [[p, not(not(q)), r], [p,not(not(q)), not(not(not(q))), r], [s]],
142         [[p, q, r], [p, q, not(q), r], [s]]
143     ).
144 test(and, [nondet]) :-

```

```

145     expand([[and(p, q)]], [[p, q]]).
146 test(not_and, [nondet]) :-
147     expand(
148         [[p, not(not(and(s, q))), and(and(not(q), p), s), r], [s]],
149         [[p, s, q, not(q), p, s, r], [s]]
150     ).
151 :- endif.

```

### Question 3B-4

Running the program with  $\neg p \wedge p \wedge \neg q$  (which is  $\neg p \rightarrow (p \rightarrow q)$  from 1.2.11c, negated and reformulated to only use NOT and AND), results in the following interaction on the console:

```

jonas@bastian:~$ swipl -g 'complete([and(not(p),and(p,not(q))))]' -t ha
lt. formulator.prolog
Complete tableau: [[not(p),p,not(q)]]
jonas@bastian:~$ gprolog --query-goal "consult('formulator.prolog'), com
plete([and(not(p),and(p,not(q))))], halt"
GNU Prolog 1.5.0 (64 bits)
Compiled Mar  2 2026, 14:18:47 with gcc
Copyright (C) 1999-2026 Daniel Diaz

| ?- consult('formulator.prolog'), complete([and(not(p),and(p,not(q))))]
, halt.
compiling /home/jonas/formulator.prolog for byte code...
/home/jonas/formulator.prolog compiled, 151 lines read - 7849 bytes writ
ten, 6 ms
Complete tableau: [[not(p),p,not(q)]]

```

Here is a screenshot of those commands (redone an hour later):

```

mc [jonas@bast PIPpropositor 9780521854337: mini1 2026.pdf mini2 2026.pdf mini3 2026.pdf Mini-project I foot Mini-proje
jonas@bastian:~$ swipl -g 'complete([and(not(p),and(p,not(q)))]). -t halt. formulator.prolog
Complete tableau: [[not(p),p,not(q)]]
jonas@bastian:~$ gprolog --query-goal "consult('formulator.prolog'), complete([and(not(p),and(p,not(q)))]), halt"
GNU Prolog 1.5.0 (64 bits)
Compiled Mar  2 2026, 14:18:47 with gcc
Copyright (C) 1999-2026 Daniel Diaz

| ?- consult('formulator.prolog'), complete([and(not(p),and(p,not(q)))]), halt.
compiling /home/jonas/formulator.prolog for byte code...
/home/jonas/formulator.prolog compiled, 151 lines read - 7749 bytes written, 6 ms
Complete tableau: [[not(p),p,not(q)]]
jonas@bastian:~$ grimshot save window
/home/jonas/2026-05-10T23:46:24.058390057+02:00.png
jonas@bastian:~$ grimshot save
active  anything area      output  screen  window
jonas@bastian:~$ grimshot save area

```

The program is missing implementation to identify and drop this contradiction, and to then conclude that it is not sound.

### Question 3B-5

Running the program on some random formula used during testing results in the following interaction on the console:

```

1 jonas@bastian:~$ swipl -g 'complete([not(and(p,and(not(not(q)),q)))]).
2 -t halt. formulator.prolog
3 Complete tableau: [[not(p),not(q),not(q)]]
4 jonas@bastian:~$ gprolog --query-goal "consult('formulator.prolog'), com
5 plete([not(and(p,and(not(not(q)),q)))]), halt"
6 GNU Prolog 1.5.0 (64 bits)
7 Compiled Mar  2 2026, 14:18:47 with gcc
8 Copyright (C) 1999-2026 Daniel Diaz
9
10 | ?- consult('formulator.prolog'), complete([not(and(p,and(not(not(q)),q
11 )))]), halt.
12 compiling /home/jonas/formulator.prolog for byte code...
13 /home/jonas/formulator.prolog compiled, 151 lines read - 7849 bytes writ
14 ten, 9 ms
15 Complete tableau: [[not(p),not(q),not(q)]]

```

Here is a screenshot of those commands (redone an hour later):

```
jonas@bastian:~$ ^Cimshot
jonas@bastian:~$ swipl -g 'complete([not(and(p,and(not(not(q)),q)))]'. -t halt. formulator.prolog
Complete tableau: [[not(p),not(q),not(q)]]
jonas@bastian:~$ gprolog --query-goal "consult('formulator.prolog'), complete([not(and(p,and(not(not(q)),q)))]), halt"
GNU Prolog 1.5.0 (64 bits)
Compiled Mar  2 2026, 14:18:47 with gcc
Copyright (C) 1999-2026 Daniel Diaz

| ?- consult('formulator.prolog'), complete([not(and(p,and(not(not(q)),q)))]), halt.
compiling /home/jonas/formulator.prolog for byte code...
/home/jonas/formulator.prolog compiled, 151 lines read - 7749 bytes written, 7 ms
Complete tableau: [[not(p),not(q),not(q)]]
jonas@bastian:~$ grimshot save area
```

Sorry, out of time :-/