Mini-project III

Jonas Smedegaard [1]

[1]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):

```
formula         = "(" formula ")" | compound | atomic;
compound        = nonequivalence | nonimplication
                | nonconjunction | nondisjunction | double negation
                | equivalence | implication | conjunction | disjunction;
nonequivalence  = NOT "(" equivalence ")";
nonimplication  = NOT "(" implication ")";
nonconjunction  = NOT "(" conjunction ")";
nondisjunction  = NOT "(" disjunction ")";
double negation = NOT NOT formula;
equivalence     = formula EQUIV formula;
implication     = formula IMPLIES formula;
conjunction     = formula AND formula;
disjunction     = formula OR formula;
EQUIV           = "⇔";
IMPLIES         = "→";
AND             = "∧";
OR              = "∨";
NOT             = "¬";
atomic          = PROPOSITIONAL | NOT PROPOSITIONAL;
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:

```
formula         = "(" formula ")" | compound | atomic;
compound        = nonequivalence | nonimplication
                | nonconjunction | nondisjunction | double negation
                | equivalence | implication | conjunction | disjunction;
nonequivalence  = NOT "(" equivalence ")";
nonimplication  = NOT "(" implication ")";
nonconjunction  = NOT "(" conjunction ")";
nondisjunction  = NOT "(" disjunction ")";
double negation = NOT NOT formula;
equivalence     = formula EQUIV formula;
implication     = formula IMPLIES formula;
conjunction     = formula AND formula;
disjunction     = formula OR formula;
EQUIV           = "⇔";
IMPLIES         = "→";
AND             = "∧";
OR              = "∨";
NOT             = "¬";
```

```
atomic          = propositional | NOT propositional;
propositional   = LETTER { LETTER } { NUMBER };
LETTER          = ? the regular expression [[:alpha:]] ?; (* unicode? *)
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 ¬ and ∧:

```
% -*- Mode: Prolog -*-
% SPDX-FileCopyrightText: 2026  Jonas Smedegaard <dr@jones.dk
% SPDX-License-Identifier: GPL-3.0-or-later
%
% Tableau-rules expansion of a propositional formula.
%
%% Usage, with REPL or non-interactively:
%% swipl -q formulator.prolog
%% swipl -g 'QUERY.' -t halt. formulator.prolog
%% gprolog --consult-file formulator.prolog
%% gprolog --query-goal "consult('formulator.prolog'), QUERY, halt"
%%
%% QUERY examples with expected response (indented):
%% complete([not(not(q))])
%%   Complete tableau: [[q]]
%% complete([not(and(p,and(not(not(q)),q)))])
%%   Complete tableau: [[not(p),not(q),not(q)]]

% log messages to console
info(String) :- format('~w~n', [String]).
info(String, Term) :- format('~w:_~w~n', [String, Term]).
warn(String) :- format('Warning:_~w~n', [String]).
warn(String, Term) :- format('Warning:_~w:_~w~n', [String, Term]).

```

```
%% first_item(Item, List)
%
% Item is the first item of List if List is non-empty
first_item([Item|_], Item).

%% distribute(Formula, TableauIn, TableauOut)
%
% TableauOut is TableauIn with Formula prepended to each branch
distribute(_, [], []).
distribute(Formula, [Rest|TableauIn], [[Formula|Rest]|TableauOut]) :-
    distribute(Formula, TableauIn, TableauOut).

%% well_formed_formula(Formula)
%
% True when Formula is a valid propositional formula
well_formed_formula(Formula) :-
    atom(Formula).
well_formed_formula(Formula) :-
    compound_formula(Formula).

%% well_formed(FormulaBranch)
%
% True when FormulaBranch is a list of well-formed formulas
well_formed([]).
well_formed([Formula|Rest]) :-
    well_formed_formula(Formula),
    well_formed(Rest).

%% compound_formula(Formula)
%
% True when Formula is a compound propositional formula
compound_formula(not(Formula)) :-
    well_formed_formula(Formula).
compound_formula(and(Formula1, Formula2)) :-
    well_formed_formula(Formula1),
    well_formed_formula(Formula2).

%% expand_branch(Formula, Branch, BranchExpanded)
%
% 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.
```

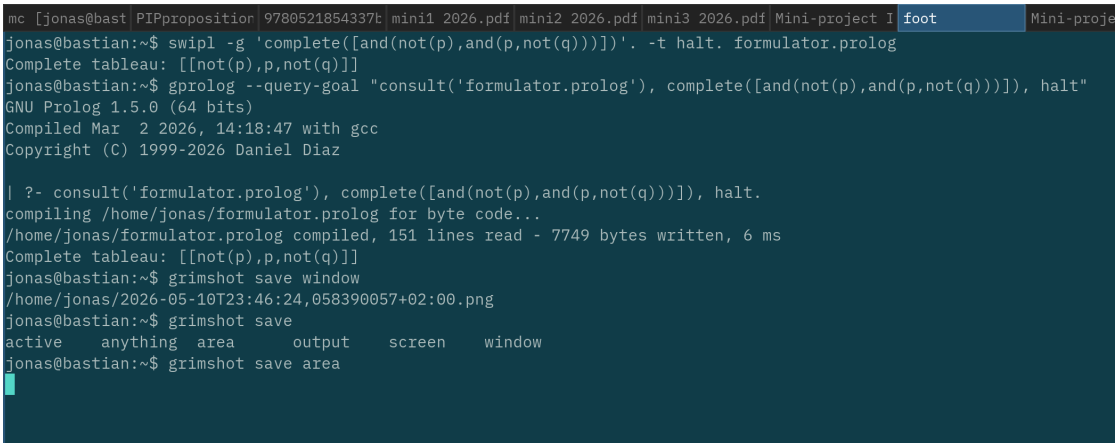
Question 3B-4

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):



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:

```
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'), com
plete([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 - 7849 bytes writ
ten, 9 ms
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 :-/