Mini-project II

Jonas Smedegaard [1]

[1] 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 $\neg((\neg p \wedge (p \to q)) \to \neg q)$, is shown below.

p	q	$\neg p$	$p \to q$	$\neg p \wedge (p \to q)$	$\neg q$	$(\neg p \wedge (p \to q)) \to \neg q$	$\neg((\neg p \wedge (p \to q)) \to \neg q)$
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 $\neg((\neg p \wedge (p \to q)) \to \neg q)$ 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 $\{p, q, \neg, \wedge, \to\}$, 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.
$v(x \to y) = 0$ 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

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

> **ℹ Correction #1**
>
> The above expression contains two errors (as pointed out in feedback):
> Error #1: The syntax confused branches and formulas.
> Error #2: Derived formulas should be negated.
> Corrected expression:
> $\{B_1, \dots, B \cup \{\phi \vee \psi\}, \dots, B_i\} \rightsquigarrow \{B_1, \dots, B \cup \{\neg\phi\}, B \cup \{\neg\psi\}, \dots, B_i\}$

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

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

> **ℹ Correction #2**
>
> The above expression contains one error (as pointed out in feedback):
> Error #1: The syntax confused branches and formulas.
> Corrected expression:
> $\{B_1, \dots, B \cup \{\phi \rightarrow \psi\}, \dots, B_i\} \rightsquigarrow \{B_1, \dots, B \cup \{\neg\phi\}, B \cup \{\psi\}, \dots, B_i\}$

Question 3

> **ℹ Addition #1**
>
> Obtaining a complete tableau by applying tableau rules can be described in pseudecode like this:

Algorithm 1 TableauCompletion

```
procedure TableauCompletion(τ)
    if τ ∈ ∅ then
        return τ                                        // tableau is closed
    repeat
        ρ ← τ
        ρ ← DropInconsistencies(ρ)
        ρ ← ApplyUnibranchRules(ρ)
    until ρ = τ                         // deplete non-branching changes first
    ρ ← ApplyBranchingRules(ρ)
    if ρ = τ then
        return ρ                              // tableau is open and complete
    TableauCompletion(ρ)
procedure DropInconsistencies(τ)
    ρ ← τ
    for all n ∈ {1, … , i ← |ρ|} where {B_1, … , B_i} ∈ ρ do
        if BranchIsInconsistent(B_n) then
            ρ ← {B_1, … , B_{n−1}, B_{n+1}, … , B_i}
    return ρ
procedure ApplyUnibranchRules(τ)
    ρ ← τ
    for all n ∈ {1, … , i ← |ρ|} where {B_1, … , B_i} ∈ ρ do
        for all m ∈ {1, … , j ← |B_n|} where {F_1, … , F_j} ∈ B_n do
            if {ϕ} ← RuledUnibranchFormula(F_m) then
                B_n ← {F_1, … , F_{m−1}, ϕ, F_{m+1}, … , F_i}
    return ρ
procedure ApplyBranchingRules(τ)
    for all n ∈ {1, … , i ← |τ|} where {B_1, … , B_i} ∈ τ do
        for all m ∈ {1, … , j ← |B_n|} where {F_1, … , F_j} ∈ B_n do
            if {ϕ, ψ} ← RuledBranchingFormulas(F_m) then
                B'_n ← {F_1, … , F_{m−1}, ϕ, F_{m+1}, … , F_i}
                B''_n ← {F_1, … , F_{m−1}, ψ, F_{m+1}, … , F_i}
                return {B_1, … , B_{n−1}, B'_n, B''_n, B_{n+1}, … , B_i}
    return τ
```

```
1:  procedure BranchIsInconsistent(B)
2:      for all n ∈ {1, … , i ← |B|} where {F_1, … , F_i} ∈ B do
3:          if |F_n| > 2 and Symbols(F_n, 1) = ¬ and Symbols(F_n, 2) ≠ ¬ then
4:              contradiction ← Symbols(F_n, 2 … |F_n|)
5:              for all m ∈ {1, … , j ← |B|} where {F_1, … , F_j} ∈ B do
6:                  if F_m = contradiction then
7:                      return true
8:      return false
9:  procedure RuledUnibranchFormula(F)
10:     if |F_n| > 2 and Symbols(F_n, 1 … 2) = ¬¬ then                // ¬¬
11:         return {Symbols(F_n, 3 … |F_n|)}
12:     else if … then
13:         …
14:     else
15:         …
16:     return ∅
17: procedure RuledBranchingFormulas(B)
18:     if … then
19:         …
20:     else if … then
21:         …
22:     else
23:         …
24:     return ∅
```

Each round of the while-loop beginning at line 2 either drops a branch, applies rules to a branch, or ends the function. Every rule reduces the length of involved formulas, which means it is certain to eventually stop with either a set of irreducible branches or the empty set of branches.

Question 4

Addition #2

$$
\begin{aligned}
\{\neg p \wedge p \wedge \neg q\} &\rightsquigarrow \{\neg p, p \wedge \neg q\} \\
&\rightsquigarrow \{\neg p, p, \neg q\}
\end{aligned}
$$

Question 5

TODO