**Exercise 1.**

a) (*new* → *good)* → *new* Is neither satisfiable or valid.

| **new** | **good** | **new → good** | **→** | **new** |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

In the first and second case the implication is not valid and in the third and fourth case the implication is valid. So the sentence is satisfiable.

b) (new → good) → (¬good → ¬new)

| new | good | new → good | → | (¬good → ¬new) |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

In all cases the sentence is valid, so the sentence is valid.

c) (red ∧ ¬round) ∨ (round → red)

| red | round | (red ∧ ¬round) | ∨ | round → red |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |

In all cases, but the second, is the sentence valid, so the sentence is satisfiable.

**Exercise 2**

1. To prove if a property does not from a knowledge base, T, you have assume that the property does not follow from T. You apply resolution calculus to T and the negation of the property, i.e. $T \cup \{\neg \varphi\}$. Working your way through the clauses, you will discover that the empty clause can not be derived. There doesn't occur a contradiction, so the negation of the property $\varphi$ can't be proven wrong, so the property does not follow from the knowledge base T.

2.

*If a Sudoku is solvable, then it is fun. If it is not solvable, then it is no fun but pastime. If a Sudoku is fun or pastime, then it is worthwhile. A Sudoku is a good game if it is worthwhile.*

s: solvable
f: fun
p: pastime
w: worthwhile
g: good game.

S: Sudoku

Knowledge base:

| | |
|---|---|
| s(S) → f(S) | ≡ ¬s(S) ∨ f(S) |
| ¬s(S) → (¬f(S) ∧ p(S)) | ≡ s(S) ∨ (¬f(S) ∧ p(S)) |
| | ≡ ( ¬f(S) ∨ s(S) ) ∧ ( p(S) ∨ s(S) ) |
| (f(S) ∨ p(S)) → w(S) | ≡ ¬( f(S) ∨ p(S) ) ∨ w(S) |
| | ≡ ( ¬f(S) ∧ ¬p(S) ) ∨ w(S) |
| | ≡ ( ¬f(S) ∨ w(S) ) ∧ ( ¬p(S) ∨ w(S) ) |
| w(S) → g(S) | ≡ ¬w(S) ∨ g(S) |

So, T = { ¬s(S) ∨ f(S), ¬f(S) ∨ s(S) , p(S) ∨ s(S),
¬f(S) ∨ w(S), ¬p(S) ∨ w(S), ¬w(S) ∨ g(S) }

i) Is a Sudoku solvable.

1. T = { ¬s(S) ∨ f(S), ¬f(S) ∨ s(S) , p(S) ∨ s(S), ¬f(S) ∨ w(S), ¬p(S) ∨ w(S), ¬w(S) ∨ g(S), ¬s(S) }

2. T = { ¬s(S) ∨ f(S), ¬f(S), p(S), ¬f(S) ∨ w(S), ¬p(S) ∨ w(S), ¬w(S) ∨ g(S), ¬s(S) }

3. T = { ¬s(S) ∨ f(S), ¬f(S), p(S), ¬f(S) ∨ w(S), w(S), ¬w(S) ∨ g(S), ¬s(S) }

4. T = { ¬s(S) ∨ f(S), ¬f(S), p(S), ¬f(S) ∨ w(S), w(S), g(S), ¬s(S) }

5. T = { ¬s(S), ¬f(S), p(S), ¬f(S) ∨ w(S), w(S), g(S), ¬s(S) }

In step 1 you make the conclusions based on ¬s(S), so ¬f(S) ∨ s(S) becomes ¬f(S) and p(S) ∨ s(S) becomes p(S). In step 2 you make the conclusions based on p(S), so ¬p(S) ∨ w(S) becomes w(S). In step 3 you make the conclusions based on w(S), so ¬w(S) ∨ g(S) becomes, g(S). In step 4 you make the conclusions based on ¬f(S), so ¬s(S) ∨ f(S) becomes ¬s(S). From there you can’t make anymore conclusions and you can’t deduce the empty clause ⊥. This means that s(S) can't be deduced from T.

ii) Is a Sudoku worthwhile.

1. $T = \{ \neg s(S) \lor f(S),\ \neg f(S) \lor s(S),\ p(S) \lor s(S),\ \neg f(S) \lor w(S),\ \neg p(S) \lor w(S),\ \neg w(S) \lor g(S),\ \neg w(S) \}$

2. $T = \{ \neg s(S) \lor f(S),\ \neg f(S) \lor s(S),\ p(S) \lor s(S),\ \neg f(S),\ \neg p(S),\ \neg w(S) \lor g(S),\ \neg w(S) \}$

3. $T = \{ \neg s(S),\ \neg f(S) \lor s(S),\ p(S) \lor s(S),\ \neg f(S),\ \neg p(S),\ \neg w(S) \lor g(S),\ \neg w(S) \}$

4. $T = \{ \neg s(S),\ \neg f(S),\ p(S),\ \neg f(S),\ \neg p(S),\ \neg w(S) \lor g(S),\ \neg w(S) \}$

5. $T = \{ \neg s(S),\ \neg f(S),\ p(S),\ \neg f(S),\ \neg p(S),\ \neg w(S) \lor g(S),\ \neg w(S) \}$

$\perp$

In step 1 you make the conclusions based on ¬w(S), so ¬f(S) ∨ w(S) becomes ¬f(S) and ¬p(S) ∨ w(S) becomes ¬p(S). In step 2 you make the conclusions based on ¬f(S), so ¬s(S) ∨ f(S) becomes ¬s(S). In step 3 you make the conclusions based on ¬s(S), so ¬f(S) ∨ s(S) becomes ¬f(S) and p(S) ∨ s(S) becomes p(S). In step 4 you can derive the empty clause based on p(S) and ¬p(S). This proves that w(S) can be derived from T.

iii) The Sudoku is a good game

1. $T = \{ \neg s(S) \lor f(S),\ \neg f(S) \lor s(S),\ p(S) \lor s(S),\ \neg f(S) \lor w(S),\ \neg p(S) \lor w(S),\ \neg w(S) \lor g(S),\ \neg g(S) \}$

2. $T = \{ \neg s(S) \lor f(S),\ \neg f(S) \lor s(S),\ p(S) \lor s(S),\ \neg f(S) \lor w(S),\ \neg p(S) \lor w(S),\ \neg w(S),\ \neg g(S) \}$

3. $T = \{ \neg s(S) \lor f(S),\ \neg f(S) \lor s(S),\ p(S) \lor s(S),\ \neg f(S),\ \neg p(S),\ \neg w(S),\ \neg g(S) \}$

4. $T = \{ \neg s(S) \lor f(S),\ \neg f(S) \lor s(S),\ s(S),\ \neg f(S),\ \neg p(S),\ \neg w(S),\ \neg g(S) \}$

5. $T = \{ f(S),\ \neg f(S) \lor s(S),\ s(S),\ \neg f(S),\ \neg p(S),\ \neg w(S),\ \neg g(S) \}$

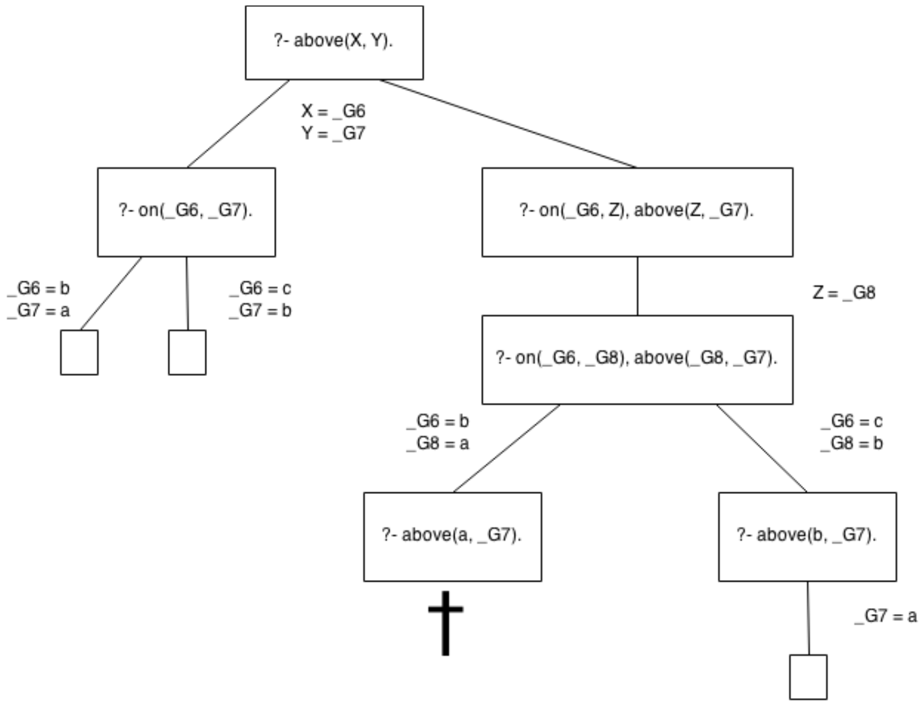
$\perp$

In step 1 we make the conclusions based on ¬g(S), so ¬w(S) ∨ g(S) becomes ¬w(S). In step 2 we make the conclusions based on ¬w(S), so ¬f(S) ∨ w(S) becomes ¬f(S) and ¬p(S) ∨ w(S) becomes ¬p(S). In step 3 we make the conclusions based on ¬p(S), so p(S) ∨ s(S) becomes s(S). In step 4 we make the conclusions based on s(S), so ¬s(S) ∨ f(S) becomes f(S). In step 5 we can deduce the empty clause from f(S) and ¬f(S), this proves that g(S) can be derived from T.

**Exercise 3**
The resolution calculus can be used to check whether a propositional formula is a tautology by applying it on the negation of the formula. If the formula is a tautology, the empty clause can be derived.

**Exercise 6**

1. X=b, Y=a; X=c, Y=b; X=c, Y=a;




On the first level prolog chooses between using the first rule or the second rule. On the left is the first rule. It has two answers X = b, Y = a and X = c, Y = b. On the right it uses the second rule. It tries to prove that there exists a block Z between X and Y, so X is on Z and Z is on Y. On the branch that closes prolog unifies b with _G6 and a with _G8. It then has to prove that a is above another block. It can't prove that, so the branch closes. On the second branch prolog chooses to unify _G6 with c and _G8 with b. It then has to prove that b is on another block, which it is, a. It unifies _G7 with a, so the answer is X = c, Y = a.

**Exercise 8**

1.

```
sun(X):-rain(X).
sun(X):-warm(Y).
warm(Y):-warm(Y).
warm(a).
rain(b).
```

The order of this program will cause a failed branch, and after that an infinite branch.

2.

```
sun(X):-rain(X).
warm(Y):-warm(Y).
```

```
sun(X):-warm(Y).
warm(a).
rain(b).
```

The order of this program will cause an infinite branch without a solution.

3.
```
sun(X):-warm(Y).
warm(a).
warm(Y):-warm(Y).
sun(X):-rain(X).
rain(b).
```

The order of this program will cause an infinite loop with successful branches.

**Exercise 9**

1. When asking prolog: ?- pred(f(X)). the program returns: X = f(X).

When asking prolog: ?- pred(f(a)). the program returns: false.

An explanation to this difference is that f(X) contains a variable (uppercase X) and f(a) does not contain a variable, so when asking prolog if pred(f(X)) which would mean: pred(X). X = f(X), then we check q(X,X) to our constant q(X,f(X)). so then q(X,f(X)). becomes: q(f(f(f( until infinity... ),f(f(f( until infinity... )) so, in order to make both of these keep filing each other in until infinity, value to be inputed as X has to be f(X), so that even though they go on infinitely, they will go on equally infinitely.

With pred(f(a)) we try to prove the implication so we get q(f(a), f(a)), which is wrong because for the implication to be true we'd need to input the input of the input of the etc... which we cannot do with f(a). Resulting in false.

2.
```
[trace] 2 ?- pred(f(X)).
Call: (6) pred(f(_G3080)) ? creep
Unify: (6) pred(f(_G3080))
Call: (7) q(f(_G3080), f(_G3080)) ? creep
Unify: (7) q(f(f(f(f(f(f(f(f(f(...))))))))),
f(f(f(f(f(f(f(f(f(...))))))))))
Exit: (7) q(f(f(f(f(f(f(f(f(f(...))))))))),
f(f(f(f(f(f(f(f(f(...)))))))))) ? creep
Exit: (6) pred(f(f(f(f(f(f(f(f(f(...)))))))))) ? creep
X = f(X).
```

Explanation:

The trace calls for pred(f(_G3080))) then the next step is to unify pred(f(_G3080)) then we call the implication of q(X, f(X)). We unify the implication, of q(X, f(X)), because we do not fail the implication because q(f(f(... infintly)), f(f(... infinitly).

Prolog then knows that X should be f(X).

```
[trace] 3 ?- pred(f(a)).
Call: (6) pred(f(a)) ? creep
Unify: (6) pred(f(a))
Call: (7) q(f(a), f(a)) ? creep
Fail: (7) q(f(a), f(a)) ? creep
Fail: (6) pred(f(a)) ? creep
```

```
false.
```

Explanation:
We call pred(f(a)) and its implication , we unify the implication
We call the q(X, f(X)).
Check the q(f(a), f(a)). which fails, because to be true it'd have to be a(f(a), f(f(a))), so we fail the rule, and we get false as result.