**Exercise 1.1** (10 points)
Consider the following CNF formulas (given in set notation) over the variable set $\mathcal{V} = \{a, b, c\}$. For each formula, mark all properties that hold. Further, for each satisfiable formula, provide a complete truth assignment over $\mathcal{V}$ that satisfies it. For each unsatisfiable formula, give a complete truth assignment over $\mathcal{V}$ that falsifies it. (Hint: See slides 17-24 of the first set of slides.)

(a) $F_2 = \{\{a, \neg b, \neg c\}, \{a, b, c\}, \{\neg a\}, \{\neg b\}, \{\neg c\}\}$ [2]
√ **unsatisfiable** √ **refutable** ◯ satisfiable ◯ valid
Assignment:
$\{a \mapsto \bot\, b \mapsto \bot\, c \mapsto \bot\}$

(b) $F_3 = \{\{a, b, \neg b\}, \{\neg a, c, \neg c\}\}$ [2]
◯ unsatisfiable ◯ refutable √ **satisfiable** √ **valid**
Assignment:
$\{a \mapsto \bot\, b \mapsto \bot\, c \mapsto \bot\}$

(c) $F_1 = \{\{a, b, \neg c\}, \{\neg a, \neg c\}, \{b, c\}\}$ [2]
◯ unsatisfiable √ **refutable** √ **satisfiable** ◯ valid
Assignment:
$\{a \mapsto \top\, b \mapsto \top\, c \mapsto \bot\}$

(d) $F_4 = \emptyset$ [2]
◯ unsatisfiable ◯ refutable √ **satisfiable** √ **valid**
Assignment:
$\{a \mapsto \bot\, b \mapsto \bot\, c \mapsto \bot\}$

(e) $F_5 = \{\emptyset\}$ [2]
√ **unsatisfiable** √ **refutable** ◯ satisfiable ◯ valid
Assignment:
$\{a \mapsto \bot\, b \mapsto \bot\, c \mapsto \bot\}$

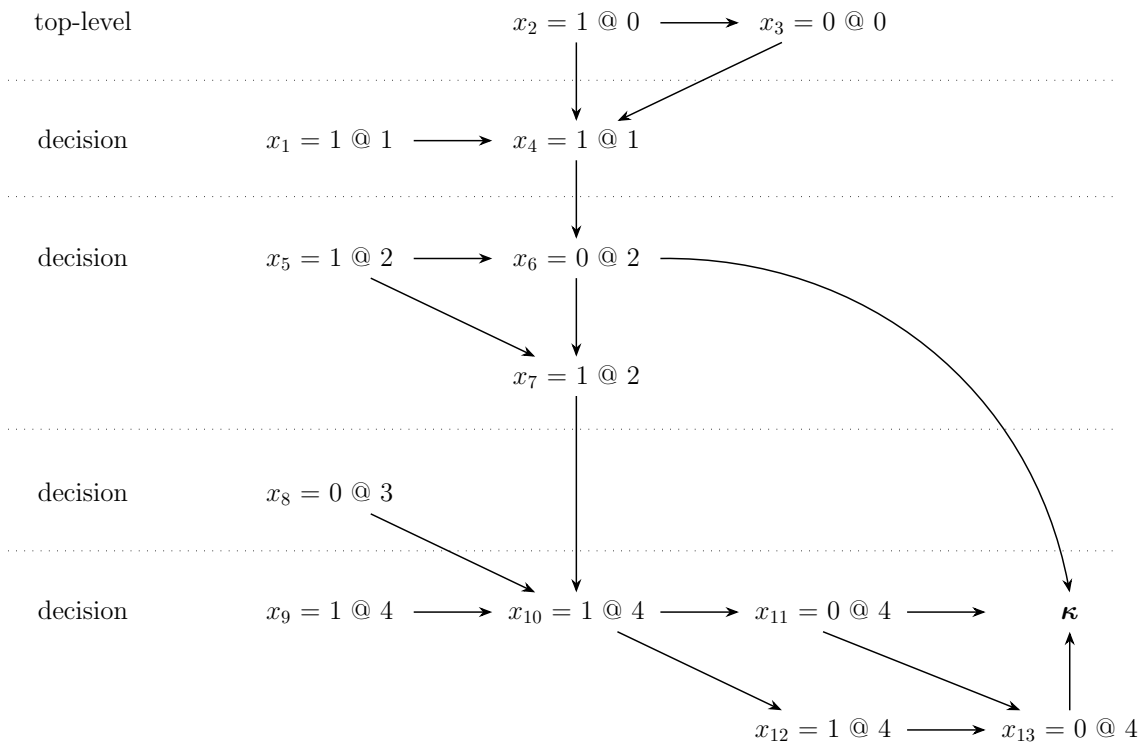
**Exercise 1.2** (20 points)
Consider the following CNF formula $F_6$:

$$(x_1)\wedge(x_1\vee x_2)\wedge(\neg x_1\vee\neg x_2)\wedge(x_2\vee\neg x_3\vee x_4)\wedge(\neg x_1\vee\neg x_3\vee\neg x_4)\wedge(\neg x_1\vee x_3\vee x_4)\wedge(x_2\vee x_3\vee\neg x_4)$$

Execute the DPLL procedure from the lecture on $F_6$ (without using the pure literal rule). Write down what you do, in particular give the resulting formula whenever Boolean Constraint Propagation is finished. Assume that the algorithm always picks $x_i$ with the smallest $i$ not yet assigned and that it always tries the positive phase of the picked variable first. (Hint: See slides 21-23 of the second set of slides.)
Esecution of the DPLL procedure, I have converted the formula into the set notation:
$F_6 := \{\{x_1\},\{x_1,x_2\},\{\neg x_1,\neg x_2\},\{x_2,\neg x_3,x_4\},\{\neg x_1,\neg x_3,\neg x_4\},\{\neg x_1,x_3,x_4\},\{x_2,x_3,\neg x_4\}\}$

1. $BCP(F_6)$
   For each unit clause it does the unit propagation. In this case it starts with one unit clause $\{x_1\}$.
   Removing all clauses where $x_1$ occurs: $\{x_1\},\{x_1,x_2\}$
   For each clause $\{\neg x_1\}\cup C$ it does unit resolution.
   New formula after resolution $F_6 := \{\neg x_2\},\{x_2,\neg x_3,x_4\},\{\neg x_3,\neg x_4\},\{x_3,x_4\},\{x_2,x_3,\neg x_4\}$.
   Another unit clause created: $\{\neg x_2\}$. With the same reasoning as in the previous step the new formula after unit propagation is: $F_6 := \{\neg x_3,x_4\},\{\neg x_3,\neg x_4\},\{x_3,x_4\},\{x_3,\neg x_4\}$
   We can see that we have no unit clause remained, so $BCP$ returns $F_6$.
2. Since $F_6 \neq \emptyset$ and $\emptyset \notin F_6$ it chooses a branching variable $x_3 \mapsto \top$ and creating $F_7 := F_6 \cup \{x_3\}$. Calling in a recursive way the procedure $DPLL(F_7)$
3. $BCP(F_7)$
   Unit propagation with the unit clause $\{x_3\}$ Removing all clause where $x_3$ occurs: $\{x_3\},\{x_3,x_4\},\{x_3,\neg x_4\}$.
   After doing unit resolution $F_7 := \{\{x_4\},\{\neg x_4\}\}$. We can see that it generates two unit clauses. Let's take $\{x_4\}$. With the same reasoning as in the previous step $F_7 := \{\emptyset\}$. So $\emptyset \in F_7$ and the recursive call returns to the parent $UNSAT$.
4. Returning to the parent it chooses $x_3 \mapsto \bot$ and creates: $F_8 := \{F_6 \cup \{\neg x_3\}\}$. Calling in a recursive way $DPLL(F_8)$.
5. $BCP(F_8)$
   Unit propagation with the unit clause $\{\neg x_3\}$ Removing all clause where $\neg x_3$ occurs: $\{\neg x_3\},\{\neg x_3,x_4\},\{\neg x_3,\neg x_4\}$.
   After doing unit resolution $F_8 := \{\{x_4\},\{\neg x_4\}\}$. Let's take $\{x_4\}$. With the same reasoning as in the previous step $F_8 := \{\emptyset\}$. So $\emptyset \in F_8$ and the recursive call returns to the parent $UNSAT$.
6. Returning to the parent it returns $UNSAT$ as well.
7. Since it is the last recursive call the procedure terminates.
8. The formula $F_6$ is $UNSAT$

**Exercise 1.3** (20 points)
Look at the following implication graph, which was constructed while solving a formula $F_7$ using the CDCL algorithm. Try to reconstruct parts of the formula from the graph and then answer all the questions below. (Hint: See slides 12-40 of the third set of slides.)




(a) Based on the implication graph, list explicitly as many clauses of $F_7$ as you can. [3]
**Answer**:
$\{x_{13}, x_6\}, \{\neg x_{13}, x_{11}, \neg x_{12}\}, \{\neg x_{11}, \neg x_{10}\}, \{x_{12}, \neg x_{10}\}, \{x_{10}, \neg x_9, \neg x_7, x_8\}$,
$\{\neg x_5, x_6, x_7\}, \{\neg x_4, \neg x_5, \neg x_6\}, \{\neg x_2, \neg x_1, x_3, x_4\}, \{x_2\}, \{\neg x_2, \neg x_3\}$

(b) What assignment is on the trail of the solver? [3]
**Answer**:
$\{x_2 \mapsto \top, x_3 \mapsto \bot, x_1 \mapsto \top, x_4 \mapsto \top, x_5 \mapsto \top, x_6 \mapsto \bot, x_7 \mapsto \top, x_8 \mapsto \bot, x_9 \mapsto \top, x_{10} \mapsto \top, x_{11} \mapsto \bot, x_{12} \mapsto \top, x_{13} \mapsto \bot, \}$

(c) Which clause of $F_7$ is falsified? [2]
**Answer**:
$\{x_{13}, x_6\}$

(d) What is the 1st UIP clause here? How did you find it? [3]
**Answer**:
1st UIP := $\{\neg x_{10}, x_6\}$ Starting from the conflict clause $\{x_{13}, x_6\}$ applying resolution in a recursive way with the clauses defined by the implication graph using an arbitrary

order for the pivot literal ( always a literal with decision level 4 ) and at each step learning a new learning clause derived by the resolution step. When I have found a learning clause with exactly one literal with decision level 4 I stopped.

(e) Give a resolution derivation of the 1st UIP clause. (Hint: use the reverse order of the variable indices if in doubt about which step to do first.) [3]
**Answer**:

1. Resolve $\{x_{13}, x_6\}$ and $\{x_{11}, \neg x_{12}, \neg x_{13}\}$:

$$\frac{\{x_{13}, x_6\}, \{x_{11}, \neg x_{12}, \neg x_{13}\}}{\{x_{11}, \neg x_{12}, x_6\}}$$

2. Resolve $\{x_{11}, \neg x_{12}, x_6\}$ and $\{\neg x_{10}, \neg x_{11}\}$:

$$\frac{\{x_{11}, \neg x_{12}, x_6\}, \{\neg x_{10}, \neg x_{11}\}}{\{\neg x_{10}, \neg x_{12}, x_6\}}$$

3. Resolve $\{\neg x_{10}, \neg x_{12}, x_6\}$ and $\{\neg x_{10}, x_{12}\}$:

$$\frac{\{\neg x_{10}, \neg x_{12}, x_6\}, \{\neg x_{10}, x_{12}\}}{\{\neg x_{10}, x_6\}}$$

4. Number of literals of decision level 4 in $C = \{\neg x_{10}, x_6\}$ equal to 1, so C is the 1st UIP.

(f) Considering the 1st UIP clause, to which decision level would the solver jump back? [3]
**Answer**: 2

(g) What would be the last UIP clause in this graph? [3]
**Answer**: It would be := $\{\neg x_9, x_8, \neg x_7, x_6\}$

**Bonus Exercise 1** (15 points (bonus))
This practical exercise is completely optional, it gives bonus points but can be ignored without any consequences.

Solve the N-Queens problem with SAT.

Implement a method QUEENS (N) that, given an argument $N$ (where $N > 3$), generates a DIMACS file that encodes the *N-Queens problem*. The problem asks whether it is possible to place $N$ chess queens on an $N \times N$ chessboard such that no two queens can attack each other (i.e., there are no two queens in the same row, column or diagonal). Try to solve your encoding of the problem with a SAT solver of your choice (for example CaDiCaL, Kissat, or MiniSat, etc.). Provide access to your source code (e.g. send it by mail or give a github link) and write a few sentences about your experiments. Preferably use a standard programming language (e.g. C, C++ or Python) to implement the DIMACS generator. Alternatively, you may consider using pySAT.

**Answer**:
You can find the solution into the file: *queen_generator.py*.
I used the python library pySAT. To run the code just write in the command line:
`./queen_generator.py`. It will show all possible solutions in a pretty way. The variables are $N^3$, where $N$ is the number of queens. Each variable $x_{ijq} = 1$ *iff* in the square in the row $i$ and column $j$ is placed the queen $q$. I left some comment for in the code explaining the variables and the constraints. I used a lot of utility functions to help me to understand whether the result is correct or not.