Exercise 5.1

\[ \varphi = (a \lor b) \ (\bar{b} \land c \land d) \ (\bar{b} \land e) \ (\bar{d} \land \bar{e} \land f) \ (\bar{a} \land g) \ (b \land \bar{g}) \ (\bar{h} \land j) \ (\bar{i} \land k) \]

order = c, f, h, i, a, b, d, e, g, h, j, k

decision level 1:

set \( c \leftarrow 0 \).

\[ \varphi = (a \lor b) \ (\bar{b} \land d) \ (\bar{b} \land e) \ (\bar{d} \land \bar{e} \land f) \ (\bar{a} \land g) \ (b \land \bar{g}) \ (\bar{h} \land j) \ (\bar{i} \land k) \]

implication graph:

\[ G = \langle \{\bar{c}\} , \{\}\rangle \]

decision level 2:

set \( f \leftarrow 0 \).

\[ \varphi = (a \lor b) \ (\bar{b} \land d) \ (\bar{b} \land e) \ (\bar{d} \land \bar{e} \land f) \ (\bar{a} \land g) \ (b \land \bar{g}) \ (\bar{h} \land j) \ (\bar{i} \land k) \]

implication graph:

\[ G = \langle \{\bar{c}, \bar{f}\} , \{\}\rangle \]

decision level 3:

set \( h \leftarrow 0 \).

\[ \varphi = (a \lor b) \ (\bar{b} \land d) \ (\bar{b} \land e) \ (\bar{d} \land \bar{e} \land f) \ (\bar{a} \land g) \ (b \land \bar{g}) \ (\bar{h} \land j) \ (\bar{i} \land k) \]

implication graph:

\[ G = \langle \{\bar{c}, \bar{f}, \bar{h}\} , \{\}\rangle \]

decision level 4:

set \( i \leftarrow 0 \).

\[ \varphi = (a \lor b) \ (\bar{b} \land d) \ (\bar{b} \land e) \ (\bar{d} \land \bar{e} \land f) \ (\bar{a} \land g) \ (b \land \bar{g}) \]

implication graph:

\[ G = \langle \{\bar{c}, \bar{f}, \bar{h}, \bar{i}\} , \{\}\rangle \]

decision level 5:

set a \leftarrow 0.

\[ \varphi = (b) \ (\bar{b} \land d) \ (\bar{b} \land e) \ (\bar{d} \land \bar{e}) \ (b \land \bar{g}) \]

implication graph:

\[ G = \langle \{\bar{c}, \bar{f}, \bar{h}, \bar{i}, \bar{a}\} , \{\}\rangle \]

assert that \( b = 1 \).

\[ \varphi = (d) \ (e) \ (\bar{d} \land \bar{e}) \]

implication graph:

\[ G = \langle \{\bar{c}, \bar{f}, \bar{h}, \bar{i}, \bar{a}, b\} , \{(\bar{a}, b)\}\rangle \]
assert that \( d = 1 \).
\[
\varphi = (e)(\bar{e})
\]
implication graph:
\[
G = \langle \{\bar{c}, \bar{f}, \bar{h}, \bar{i}, \bar{a}, b, d\} , \{(\bar{a}, b) , (\bar{c}, d) , (b, d)\}\rangle
\]
assert that \( e = 0 \) and \( e = 1 \).
\[
\varphi = \text{Impossible}
\]
implication graph:
\[
G = \langle \{\bar{c}, \bar{f}, \bar{h}, \bar{i}, \bar{a}, b, d, e, \bar{e}\} , \{(\bar{a}, b) , (\bar{c}, d) , (b, d) , (b, e) , (d, \bar{e}) , (\bar{f}, \bar{e})\}\rangle
\]
find the cut to create the new phrase:
\[
S = \{e, \bar{e}\}
\]
\[
cut = \{(b, e) , (d, \bar{e}) , (\bar{f}, \bar{e})\}
\]
in \( \{b, d, \bar{f}\} \) \( b \) and \( d \) are from the last level.
\[
S = \{e, \bar{e}, d\}
\]
\[
cut = \{(b, e) , (\bar{f}, \bar{e}) , (b, d) , (\bar{e}, d)\}
\]
in \( \{b, \bar{c}, \bar{f}\} \) only \( b \) is from the last level.
therefore we add the phrase: \( \bar{b} \lor c \lor f \) and return to decision level 2:
\[
\varphi = (a \lor b) (\bar{b} \lor d) (\bar{b} \lor e) (\bar{d} \lor \bar{e}) (\bar{a} \lor g) (b \lor \bar{g}) (\bar{h} \lor j) (\bar{i} \lor k) (\bar{b} \lor c \lor f) =
\]
\[
= (a \lor b) (\bar{b} \lor d) (\bar{b} \lor e) (\bar{d} \lor \bar{e}) (\bar{a} \lor g) (b \lor \bar{g}) (\bar{h} \lor j) (\bar{i} \lor k) (\bar{b})
\]
implication graph:
\[
G = \langle \{\bar{c}, \bar{f}\} , \{\}\rangle
\]
assert that \( b = 0 \).
\[
\varphi = (a) (\bar{d} \lor \bar{e}) (\bar{a} \lor g) (\bar{g}) (\bar{h} \lor j) (\bar{i} \lor k)
\]
implication graph:
\[
G = \langle \{\bar{c}, \bar{f}, \bar{b}\} , \{(\bar{c}, \bar{b}) , (\bar{f}, \bar{b})\}\rangle
\]
assert that \( a = 1 \).
\[
\varphi = (\bar{d} \lor \bar{e}) (g) (\bar{g}) (\bar{h} \lor j) (\bar{i} \lor k)
\]
implication graph:
\[
G = \langle \{\bar{c}, \bar{f}, \bar{b}, a\} , \{(\bar{c}, \bar{b}) , (\bar{f}, \bar{b}) , (\bar{b}, a)\}\rangle
\]
assert that \( g = 0 \) and \( g = 1 \).
\[
\varphi = \text{Impossible}
\]
implication graph:
\[
G = \langle \{\bar{c}, \bar{f}, b, a, g, \bar{g}\} , \{(\bar{c}, \bar{b}) , (\bar{f}, \bar{b}) , (\bar{b}, \bar{a}) , (\bar{b}, \bar{g}) , (a, g)\}\rangle
\]
find the cut to create the new phrase:
\[
S = \{g, \bar{g}\}
\]
\[
cut = \{(\bar{b}, \bar{g}) , (a, g)\}
\]
in \( \{\bar{b}, a\} \) \( \bar{b} \) and \( a \) are from the last level.
\[
S = \{g, \bar{g}, a\}
\]
\[
cut = \{(\bar{b}, \bar{g}) , (\bar{b}, a)\}
\]
in \( \{\bar{b}\} \) only \( \bar{b} \) is from the last level.
therefore we add the phrase: \( b \) and return to decision level 0:
\[
\varphi = (a \lor b) (\bar{b} \lor c \lor d) (\bar{b} \lor e) (\bar{d} \lor \bar{e} \lor f) (\bar{a} \lor g) (b \lor \bar{g}) (\bar{h} \lor j) (\bar{i} \lor k) (\bar{b} \lor c \lor f) (b)
\]
implication graph:
\[
G = \langle \{\} , \{\}\rangle
\]
assert that \( b = 1 \).
\[
\varphi = (c \lor d) (e) (\bar{d} \lor \bar{e} \lor f) (\bar{a} \lor g) (\bar{h} \lor j) (\bar{i} \lor k) (c \lor f)
\]
implication graph:
\[ G = \langle \{b\}, \{\} \rangle \]
assert that \( e = 1 \).
\[ \varphi = (c \lor d) (d \lor b) (\overline{a} \lor g) (h \lor j) (i \lor k) (c \lor f) \]
implication graph:
\[ G = \langle \{b, e\}, \{(b, e)\} \rangle \]
decision level 1:
set \( c \leftarrow 0 \).
\[ \varphi = (d) (d \lor f) (\overline{a} \lor g) (h \lor j) (i \lor k) (f) \]
implication graph:
\[ G = \langle \{b, e, \overline{c}\}, \{(b, e)\} \rangle \]
assert that \( f = 1 \).
\[ \varphi = (d) (\overline{a} \lor g) (h \lor j) (i \lor k) \]
implication graph:
\[ G = \langle \{b, e, \overline{c}, f\}, \{(b, e), (b, f), (\overline{c}, f)\} \rangle \]
assert that \( d = 1 \).
\[ \varphi = (\overline{a} \lor g) (h \lor j) (i \lor k) \]
implication graph:
\[ G = \langle \{b, e, \overline{c}, f, d\}, \{(b, e), (b, f), (\overline{c}, f), (b, d), (\overline{c}, d)\} \rangle \]
decision level 2:
set \( h \leftarrow 0 \).
\[ \varphi = (\overline{a} \lor g) (i \lor k) \]
implication graph:
\[ G = \langle \{b, e, \overline{c}, f, d, h\}, \{(b, e), (b, f), (\overline{c}, f), (b, d), (\overline{c}, d)\} \rangle \]
decision level 3:
set \( i \leftarrow 0 \).
\[ \varphi = (\overline{a} \lor g) \]
implication graph:
\[ G = \langle \{b, e, \overline{c}, f, d, h, i\}, \{(b, e), (b, f), (\overline{c}, f), (b, d), (\overline{c}, d)\} \rangle \]
decision level 4:
set \( a \leftarrow 0 \).
\[ \varphi = 1 \]
implication graph:
\[ G = \langle \{b, e, \overline{c}, f, d, h, i, a\}, \{(b, e), (b, f), (\overline{c}, f), (b, d), (\overline{c}, d)\} \rangle \]
decision level 5:
set \( g \leftarrow 0 \).
\[ \varphi = 1 \]
implication graph:
\[ G = \langle \{b, e, \overline{c}, f, d, h, i, a, g\}, \{(b, e), (b, f), (\overline{c}, f), (b, d), (\overline{c}, d)\} \rangle \]
decision level 6:
set \( j \leftarrow 0 \).
\[ \varphi = 1 \]
implication graph:
\[ G = \langle \{b, e, \overline{c}, f, d, h, i, a, g, j\}, \{(b, e), (b, f), (\overline{c}, f), (b, d), (\overline{c}, d)\} \rangle \]
decision level 7:
set \( k \leftarrow 0 \).
\( \varphi = 1 \)

implication graph:
\[ G = \langle \{b, e, \bar{c}, f, d, \bar{h}, \bar{t}, \bar{a}, \bar{g}, \bar{j}, \bar{k}\}, \{(b, e), (b, f), (\bar{c}, f), (b, d), (\bar{c}, d)\} \rangle \]

We have found that the formula is satisfiable, with:
\[ a = 0 \]
\[ b = 1 \]
\[ c = 0 \]
\[ d = 1 \]
\[ e = 1 \]
\[ f = 1 \]
\[ g = 0 \]
\[ h = 0 \]
\[ i = 0 \]
\[ j = 0 \]
\[ k = 0 \]

**Exercise 5.2**

Base: a formula with a single variable \( x \) will be unsatisfiable only if the formula is \( (x)(\bar{x}) \).

by a single resolution \( (x \lor ()) (\bar{x} \lor ()) \Rightarrow (()) \lor () = () \).

Step: assume that for an unsatisfiable formula with \( n \) variables, we can derive the empty clause by a sequence of resolution steps. and show that we can do the same for an unsatisfiable formula with \( n + 1 \) variables.

Let \( F \) be a formula with \( n + 1 \) parameters, and let us take some variable \( x \) from it.

Let us divide \( F \) to two parts \( F_x \): all the phrases that do not contain \( x \) and \( F_{\bar{x}} \): all the phrases that do not contain \( \bar{x} \). note that \( F_x \) has the same phrases as \( F[x = 1] \) aside from the occasional \( \bar{x} \) appearing in some of the phrases of \( F_x \) (all the phrases containing \( x \) in \( F \) will already be satisfied and will not appear). similarly note that \( F_{\bar{x}} \) has the same phrases as \( F[x = 0] \) aside from the occasional \( x \) appearing in some of the phrases in \( F_{\bar{x}} \).

According to the induction assumption we can derive the empty clause from \( F[x = 0] \) and \( F[x = 1] \) using some sequence of resolutions, since these has only \( n \) variables and it cannot be satisfiable since otherwise \( F \) is satisfiable too.

Performing the same sequence of resolution steps on \( F_x \) that we would have performed on \( F[x = 1] \) would be possible, since the resolution step \( (C \lor \ell) (C' \lor \bar{\ell}) = (C \lor C') \) does not depend on the values of the clauses, so we can ignore the literal \( x \), but the created clause may contain \( x \) instead of the empty clause. if we got the empty clause we are done, otherwise we created the clause \( (x) \).

Performing the same sequence of resolution steps on \( F_{\bar{x}} \) that we would have performed on \( F[x = 0] \) would be possible, since the resolution step \( (C \lor \ell) (C' \lor \bar{\ell}) = (C \lor C') \) does not depend on the values of the clauses, so we can ignore the lit-
eral $\overline{x}$, but the created clause may contain $\overline{x}$ instead of the empty clause, if we got the empty clause we are done, otherwise we created the clause ($\overline{x}$).

We can create $(x) (\overline{x})$ and from it we can resolve $(x \lor (\overline{x} \lor ())) \Rightarrow ((\lor ())) = ()$.

Therefore, we have shown that for an unsatisfiable formula of size $n + 1$ we can derive the empty clause by a sequence of resolution steps.

Therefore we have proven the requested by induction.

Exercise 5.3

first, let us assume that all the formulas phrases are of size 2.

We may assume so by performing the following action for every phrase of size one:

1. assign a value to the variable in the phrase so that the phrase will be satisfied.

   (a) if it is already assigned, return that the formula is unsatisfiable.

2. if there are still phrases of size one (if they were created by previous assignments as well) return to step 1.

For the formula $F = (a_1 \lor b_1) (a_2 \lor b_2) \ldots (a_n \lor b_n)$ on $m$ variables $v_i$ for $i \in [m]$, and $a_i, b_i, c_i \in \{v_j : j \in [m]\} \cup \{\overline{v_j} : j \in [m]\}$ build the following directed graph:

$$G = \langle V, E \rangle$$

$V = \{v_j : j \in [m]\} \cup \{\overline{v_j} : j \in [m]\}$

$E = \{(a_i, b_i), (b_i, c_i) : j \in [n]\}$

(this is basically the implication graph)

1. find the connected components of the graph (can be done in linear time using DFS).

2. for each connected component:

   (a) check if there is a variable and its negation in the connected component

      i. if there is return that the formula is unsatisfiable.

3. created the connected component graph.

4. find a topological order for the connected components.

5. iterate over the connected components in reverse topological order

   (a) for every component with variable not set already, set all the variables in it according to the literals in the connected component.
6. return the assignment created.

Runtime:
the first part of the algorithm (assigning single literal phrases), may take $O(n \cdot m)$, since we may remove at most $m$ single phrase parameters (one for every variable) and the removal will take at most $n$ time (removing the variable from all phrases).

finding connected components takes $O(m + n)$
iterating over all pairs of literals in the connected component takes $O(m^2)$
creating the connected component graph will take only $O(m + n)$, by simply iterating over the edges to create the new graph.
finding a topological order can be done using DFS $O(m + n)$.
iterating over the graph and setting the variables is linear as well.
total runtime is $O(n \cdot m + m^2 + m + n) = O(n^2)$ (there are $O(n)$ variables, since there are only $2n$ spaces for them in $m$ phrases)

Proof:
The first phase of the algorithm, the values set to variables must be so at every satisfying assignment. therefore if we got a conflict then it means the formula is not satisfiable.

If there are both a variable and its negation in a connected component, then, since one of them must be true at an assignment, let us look at the path from $x$ (the true literal) to $y$ (the false literal), there must be one since this is a connected component, let us show by induction that every literal in the path must be true:

base: $x$ is set.

step: the path until $z$ is true let us show that $w$ that is immediatly after $z$ in the path is true as well. $(z, w) \in E$ therefore $(\bar{z} \lor w)$ is in $F$, and since $z$ is true $(\bar{z} \lor w) = (w)$ $w$ must be true as well.

Therefore both the variable and its negation must be true - contradiction!
the graph is not satisfiable.

We won't get a contradiction during the assignment of the connected components in the reverse topological order, since if we do get a contradiction, we have set a variable in a lower topographical order, so it must have been set somehow, and it can only be set by lower topographical order - therefore we have a cycle - but we can't have one - therefore the assignment would be legal.

Therefore we have found an algorithm with sufficiently low runtime.

Exercise 5.4

Let us look at the step when we create a clause, we have got there from the situation where $x$ and $\bar{x}$ are in the implication graph.

Let us show that every clause is created in a series of resolution steps.

When creating a clause, first, we run into a contradiction, meaning both $x$ and $\bar{x}$ are in the implication graph.
Let us show by induction that every clause we create after $i$ moves of the cut it is created from a series of resolution steps.

Base: $i = 0$, since $x$ and $\overline{x}$ are both in the implication graph, the clauses that contained them $(C \lor x)$ and $(C' \lor \overline{x})$, such that for every $c \in C \cup C' \overline{c}$ is in the implication graph with an edge to $x$ or $\overline{x}$, therefore the phrase is $(C \lor C')$ which is exactly a resolution step.

Step: assume that after $i$ moves of the cut, the clause we created is created as a sequence of resolution steps, and show that the new phrase is created as a sequence of resolution steps.

The node that is moved between the sides of the cut which is the literal $\ell$, it must have an edge in the implication graph to the other side of the cut, therefore the current clause is of the form $(\ell \lor C)$. and since $\ell$ is in the implication graph (and it is not a guess, otherwise it would be the only one within the decision level, and we would have already found out the clause for this step of the algorithm) and therefore it is learned from some clause $(\ell \lor C')$ such that for all $c \in C' \overline{c}$ is in the implication graph with an edge to $\ell$ therefore the new vertices are the old vertex without $\ell$ but with these $\overline{c}$s, and therefore the clause we learn is exactly $(C \lor C')$ that is a resolution step from $(\ell \lor C)$ and $(\ell \lor C')$.

Therefore every clause we create we perform as a series of resolution steps we actually perform. therefore the algorithm will take at least as much time as the minimal number of resolution steps, and according to the theorem, in the case of $PH_n$ it is at least $2^{\frac{n}{2 \log n}} = O \left( 2^{\frac{n}{2 \log n}} \right)$ time.

**Exercise 5.5**

(a)

Since we assume the formula is satisfiable, therefore there is some satisfying assignment $A_{sat}$.

Let us define $w(A)$ - the number of values different between $A$ and $A_{sat}$.

Now let us look at value of $w(A_t)$ where $A_t$ is the assignment after iteration $t$ of the algorithm.

if we did not return that the formula is satisfiable before iteration $t$, there is some clause $C$ that is unsatisfied, and therefore $A_{t-1} \neq A_{sat}$ (otherwise all clauses would have been satisfied). in the (at most) two literals in $C$, at least one must have a different value in $A_{t-1}$ then in $A_{sat}$ (since otherwise it would have been satisfied). Therefore, the probability to flip the value of a literal that $A_{t-1}$ and $A_{sat}$ don’t agree about is at least half, otherwise we flip the value of a variable they both agreed on.

Therefore $w(A_t) = w(A_{t-1}) - 1$ with probability $\geq \frac{1}{2}$, and $w(A_t) = w(A_{t-1}) + 1$ with probability $\leq \frac{1}{2}$.

define $p = P(w(A_t) = w(A_{t-1}) - 1) \geq \frac{1}{2}$

define $q = P(w(A_t) = w(A_{t-1}) + 1) = 1 - P(w(A_t) = w(A_{t-1}) - 1) = 1 - p.$
Let us define $h_j$ as the expected number of steps left for reaching $A_{s_{at}}$ with $w(A_t) = j$.

If $j = 0$ we have already finished the algorithm and therefore $h_0 = 0$.

If $j = n$ every flip will be reducing the distance and therefore $h_n = h_{n-1} + 1$.

else:

$h_j = p \cdot (h_{j-1} + 1) + q \cdot (h_{j+1} + 1) = p \cdot (h_{j-1} + 1) + (1 - p) \cdot (h_{j+1} + 1) = p \cdot h_{j-1} + p \cdot h_{j+1} + 1 - p \cdot h_{j+1} - p = p \cdot (h_{j-1} - h_{j+1}) + h_{j+1} + 1 \leq h_{j-1} \leq h_{j+1}$ (since one step moves will pass through the other state, so the expectancy must be more steps)

and since $p \geq \frac{1}{2}$

$\leq \frac{1}{2} \cdot (h_{j-1} - h_{j+1}) + h_{j+1} + 1 = \frac{1}{2} \cdot (h_{j-1} + h_{j+1}) + 1$

$h_j \leq \frac{1}{2} \cdot (h_{j-1} + h_{j+1}) + 1$

Let us prove by induction that for $1 \leq j \leq n - 1$: $h_j \leq \frac{j}{j+1} h_{j+1} + j$.

Base: $j = 1$, $h_1 \leq \frac{1}{2} \cdot (h_0 + h_2) + 1 = \frac{1}{2} \cdot h_2 + 1$.

Step: assume for $j$ that $h_j \leq \frac{j}{j+1} h_{j+1} + j$ prove that for $j + 1 < n$, $h_{j+1} \leq \frac{j+1}{j+2} h_{j+2} + j + 1$.

$h_{j+1} \leq \frac{1}{2} \cdot (h_{j-1} + h_{j+1}) + 1 \leq \frac{1}{2} \cdot \left( \frac{j}{j+1} h_{j+1} + j + h_{j+2} \right) + 1$

$2h_{j+1} \leq \frac{j}{j+1} h_{j+1} + h_{j+2} + j + 2 \Rightarrow \frac{j+2}{j+1} h_{j+1} \leq h_{j+2} + j + 2 \Rightarrow (j + 1) \cdot h_{j+2} + (j + 1) \cdot (j + 2)$

$h_{j+1} \leq \frac{j+1}{j+2} h_{j+2} + j + 1$.

Therefore we have proved by induction that $h_j \leq \frac{j}{j+1} h_{j+1} + j$.

and therefore $h_{n-1} \leq \frac{n-1}{n} h_n + n - 1$, and since $h_n = h_{n-1} + 1$.

$h_{n-1} \leq \frac{n-1}{n} (h_{n+1} - 1) + n - 1 \Rightarrow n \cdot h_{n-1} \leq (n - 1) \cdot (h_{n-1} + 1) + n - n \Rightarrow$

$\Rightarrow h_{n-1} \leq n - 1 + n^2 - n = n^2 - 1$.

$h_n = h_{n-1} + 1 \leq n^2$.

and since for every $j < n$, $h_j < h_n$ the bound for the expectancy of the number of steps for every initial assignment is $n^2$.

(b)

Let us look at the random variable $X$ - the number of steps from the beginning of the algorithm until we find a satisfying assignment.

according to the markov inequality:

$P (X \geq 2n^2) \leq \frac{E(X)}{2n^2} \leq \frac{n^2}{2n^2} = \frac{1}{2}$

Therefore if the algorithm succeeds in finding an assignment in case the formula is satisfiable is $1 - P (X \geq 2n^2) \geq 1 - \frac{1}{2} = \frac{1}{2}$

(c)

another more efficient option is performing $2k \cdot n^2$ iterations. according to our analysis in (b) it does not matter what is our original assignment, the probability of reaching a satisfying assignment after $2n^2$ steps is greater than $\frac{1}{2}$. therefore, we can devide all the iterations into $k$ independent sets of iterations (aside from
the starting point that we noted does not really matter) and in the case we have failed we know that the probability that the formula is actually satisfiable is smaller than \((1 - \frac{1}{2})^k = \frac{1}{2^k}\).