# Knowledge Representation and Reasoning 2023

## Homework assignment #1

Didier Merk (11037172), Tsatsral Mendsuren (14530775)

**Exercise 1.** - *CDCL Algorithm*

$$
\begin{array}{rll}
\varphi = & (\neg p_1 \vee \neg p_4 \vee p_5) \wedge & \text{(clause } c_1\text{)} ✓ \\
& (\neg p_5 \vee p_6) \wedge & \text{(clause } c_2\text{)} ✓ \\
& (\neg p_5 \vee p_7) \wedge & \text{(clause } c_3\text{)} ✓ \\
& (\neg p_6 \vee \neg p_7 \vee p_8) \wedge & \text{(clause } c_4\text{)} ✓ \\
& (\neg p_8 \vee p_9) \wedge & \text{(clause } c_5\text{)} ✓ \\
& (\neg p_1 \vee \neg p_8 \vee \neg p_9) \wedge & \text{(clause } c_6\text{)} ✗ \\
& (p_2 \vee p_3) & \text{(clause } c_7\text{)} ✓
\end{array}
$$

(a) The partial assignment that the CDCL algorithm has in memory at the moment when it runs into the first conflict is: $p_1$, $p_2$, $p_3$, $p_4$, $\underline{p_5}$, $\underline{p_6}$, $\underline{p_7}$, $\underline{p_8}$, $\underline{p_9}$. The conflict is demonstrated above.

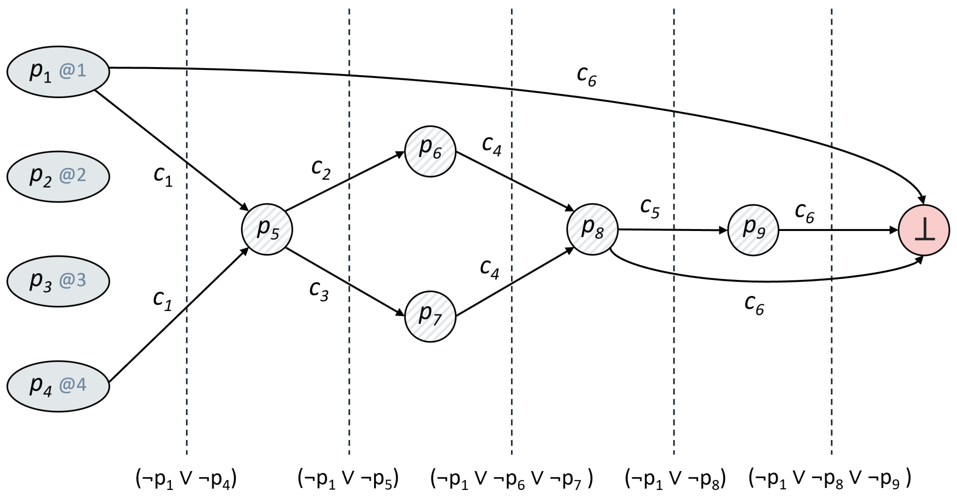
To get to this answer we follow the steps of the CDCL algorithm, where first we set $p_1$ to true. This does not lead to any unit propagation, and we can decide to set $p_2$ to true. Since this still does not lead to unit propagation, we can decide to set $p_3$ to true. Again, there is no unit propagation, meaning we can choose to also set $p_4$ to true.

This is where we have our first unit propagation, we have set $p_1$ and $p_4$ to true, which according to clause $c_1$ means that $p_5$ must be true. This in turn means that $p_6$ and $p_7$ must be true according to clauses $c_2$ and $c_3$ respectively. According to clause $c_4$ we now propagate $p_8$ to be true, which leads to the fact that $p_9$ must be true (clause $c_5$).

We now run into our first conflict; $p_1$, $p_8$ and $p_9$ are set to true, however one of them needs to be false according to clause $c_6$.

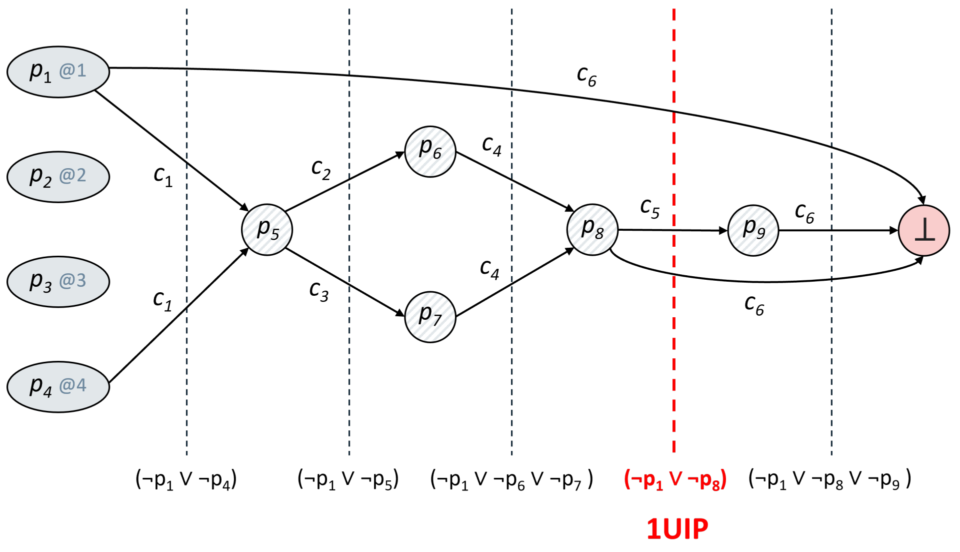
To summarize, $p_1$, $p_2$, $p_3$ and $p_4$ are the results of decisions made; $\underline{p_5}$, $\underline{p_6}$, $\underline{p_7}$, $\underline{p_8}$ and $\underline{p_9}$ are the result of propagation. This is when the algorithm runs into its first conflict (clause $c_6$).

(b) See the figure below:



In this figure we demonstrate the conflict graph for the state of the algorithm when it runs into its first conflict. The '@1', '@2', '@3' and '@4' demonstrate the decision levels. As described in question a, after the fourth decision ($p_4$ set to true), we propagate until we run into a conflict. Next to each arrow the clause that is involved in the unit propagation is given.

(c) The first clause learned using the 1UIP scheme is: $(\neg p_1 \lor \neg p_8)$, see the figure below:



$p_1$ @1
$p_2$ @2
$p_3$ @3
$p_4$ @4
$c_1$
$c_1$
$c_2$
$c_3$
$c_4$
$c_4$
$c_5$
$c_6$
$c_6$
$c_6$
$p_5$
$p_6$
$p_7$
$p_8$
$p_9$
$\perp$
$(\neg p_1 \lor \neg p_4)$
$(\neg p_1 \lor \neg p_5)$
$(\neg p_1 \lor \neg p_6 \lor \neg p_7)$
$(\neg p_1 \lor \neg p_8)$
$(\neg p_1 \lor \neg p_8 \lor \neg p_9)$
1UIP

The cut is constructed by putting all literals that don't have the highest decision level left (L) of the cut, which are $p_1$, $p_2$ and $p_3$. We call the decision literal of the highest level $\ell_d$, which is $p_4$. The next step is to find the literal closest to the conflict ($\perp$), that lies on all paths from $\ell_d$ ($p_4$), to the conflict $\perp$. Both $p_5$ and $p_8$ lie on all paths from $p_4$ to $\perp$, however $p_8$ is the one closest to the conflict; we call it $\ell_1$.

We then put $\ell_d$, $\ell_1$ and all literals in between also on the left side of the 1UIP cut. The remaining literals (and $\perp$) will go on the right side of the cut, as demonstrated in the figure above. The clause learned, $(\neg p_1 \lor \neg p_8)$, is the clause corresponding to the 1UIP cut.

(d) The algorithm can directly backjump 3 decision levels. In the CDCL algorithm, as explained on slide 26 of lecture 3, the algorithm backjumps to (the negation of) the literal in the 1UIP clause that has the second highest-decision level.

In our case, the 1UIP clause is $(\neg p_1 \lor \neg p_8)$, where $p_1$ is on decision level 1 and $p_8$ is on decision level 4 (as explained in question 1.b). This means that the second highest decision level in the 1UIP clause is 1. We were on decision level 4 and immediately backjump to level 1, meaning we are able to backjump 3 levels.

The fact we can jump this many levels in mainly because $p_2$ and $p_3$, as shown in the figures in questions 1.b and 1.c are 'isolated' nodes in the conflict graph. They are not in the 1UIP clause learned, meaning that we can jump all the way back to decision level 1.

---

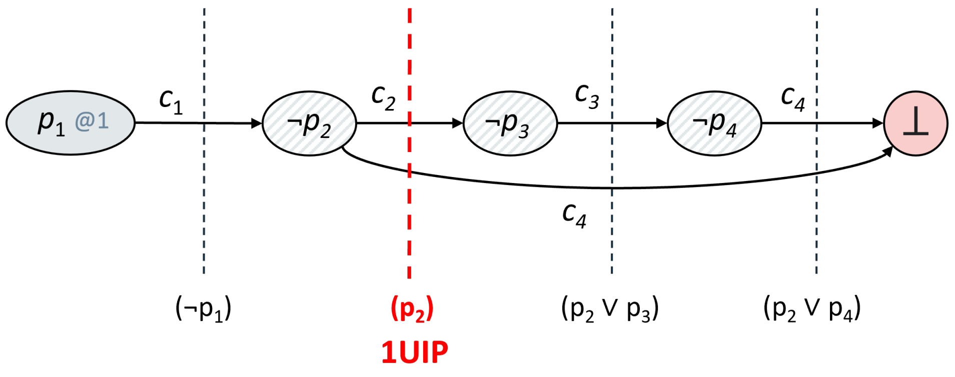
**Exercise 2.** - *CDCL on 2CNF formulas*

Statement: When the CDCL algorithm with the 1UIP learning scheme is run on a 2CNF formula, the clauses that are learned by the algorithm are unit clauses.

To illustrate this better, the conflict graph of the example CNF formula in the question is demonstrated. For simplicity reasons clause $c_5$ and $c_6$ have been left out. This illustration is only to intuitively understand what we are saying, it should not be interpreted as a generalisation.

$$
\begin{array}{lll}
\varphi = & (\neg p_1 \lor \neg p_2) \land & \text{(clause } c_1\text{)} \checkmark \\
& (p_2 \lor \neg p_3) \land & \text{(clause } c_2\text{)} \checkmark \\
& (p_3 \lor \neg p_4) \land & \text{(clause } c_3\text{)} \checkmark \\
& (p_2 \lor p_4) & \text{(clause } c_4\text{)} ✗
\end{array}
$$

Partial assignment in memory: $p_1$, $\underline{\neg p_2}$, $\underline{\neg p_3}$, $\underline{\neg p_4}$. The conflict graph looks as follows:



(a) It is stated that when a CDCL algorithm with a 1UIP learning scheme is ran on a 2CNF formula, the clauses learned by the algorithm are always unit clauses. In the example above for example it is shown that the 1UIP clause learned is $p_2$, which is a unit clause. In this question we will prove that this is generally always true.

**(1)** The first thing to note about 2CNF formulas is that all paths going to the conflict, will originate from the highest decision literal $\ell_d$ (in the example above $\ell_d = p_1$). Since clauses in a 2CNF can only consists of at most two literals, meaning that when a decision is made it will always result into a 'domino effect' of propagations until a conflict is reached, or the formula is found to be satisfied. To explain it more intuitively: lower decision levels will never be part of a conflict, since their propagation domino effect should have already resulted in either satisfaction or conflict.

**(2)** From this first fact, it follows that the $\ell_1$ literal, which in the 1UIP scheme is the literal closest to the conflict $\perp$, that lies on all paths from the highest decision literal $\ell_d$ to the conflict $\perp$, will always be the only literal that has arrows in the conflict graph that 'go through' the 1UIP cut. This is demonstrated visually above, where the $\ell_1$ literal $\neg p_2$ is the only literal that has arrows going through the 1UIP cut.

To reiterate: (2) is true because *(a)* by definition there will be no literals from lower decision levels that play a part in the conflict and *(b)* there will also by definition be only one literal from the highest decision level cutting through the 1UIP cut, because if there were multiple, it wouldn't be the correct 1UIP cut.

These two things **(1)** and **(2)** together, mean that all the arrows going through the 1UIP cut come from *only one literal*, meaning you the clause learned is always a *unit* clause.

(b) The statement proven in question 2.a can be used to show that the CDCL algorithm with the 1UIP learning scheme runs efficiently on 2CNF formulas, with the number of decisions and number of conflicts encountered are both bounded by $n^2$ with $n$ as the number of propositional variables.

First, the number of decisions the CDCL algorithm needs to make on 2CNF formulas is bounded by $\mathcal{O}(n^2)$ in the worst case for an unsolvable 2CNF formula. In the worst case, initially, we need to assign at most $n$ propositional variables before a conflict is encountered, that is we need to make at most $n$ decisions. Then we need to learn a new 1UIP clause which will be a unit clause for 2CNF formula and we backjump to decision level 0 (because there is only one literal in the clause and the most recent point where the new 1UIP clause enables unit propagation of a previously unpropagated literal). After the backjump, we again need to make $n-1$ decisions in the worst case before a conflict is encountered. The reason why we can make one less decision is because we've already excluded one decision that will always lead to conflict by 1UIP learning. In the third trial, we make $n-2$ decisions in the worst case; in the fourth trial, we make $n-3$ decisions, etcetera. Continuing this process will lead to a total of $\mathcal{O}(n^2)$ decisions.

Second, the number of conflicts is also bounded by $\mathcal{O}(n^2)$. One reason for this is because the number of conflicts must not exceed the number of decisions, because one decision can at most lead to one conflict. Since we've also proved that the number of decisions is bounded by $\mathcal{O}(n^2)$, this follows that the number of conflicts is also bounded by $\mathcal{O}(n^2)$.

When one new decision is made or when one conflict is encountered, we know the number of the corresponding updates and operations the CDCL algorithm with 1UIP learning scheme need to do is only linearly proportional to the number of decisions and conflicts. Given that the number of conflicts and decisions is bounded by $\mathcal{O}(n^2)$, it follows that the total complexity of this algorithm is only polynomial. Hence, the CDCL algorithm with 1UIP learning scheme is NP-complete on 2CNF formulas, which explains why it runs efficiently on 2CNF formulas.

---

**Exercise 3.** - *Edge Coloring Graphs*

Wikipedia describes **edge coloring** of a graph as *"the assignment of colors to each edge of a graph, such that no two adjacent edges have the same color"*, which is called a 'proper' color assignment. In this exercise we will describe how this problem of edge coloring of graphs can be solved, using propositional logic and an SAT solving algorithm. That is, we can find a proper edge coloring of an arbitrary input graph G, using a *minimal* amount of colors.

To do this, the first thing we need to do is 'translate' the problem into something that can be solved by an SAT solving algorithm. We want to write the problem into the language of propositional logic, in conjunctive normal form (CNF) to be precise. As this is what an SAT solver is able to solve.

First we define the input of the problem:

1. The input is a finite undirected graph G = (V, E), where V is the set of vertices, and E $\subseteq$ V x V is a set of undirected edges.
2. An *edge coloring* is a mapping $\mu$ : E $\rightarrow \mathbb{N}$, which assigns an edge $e \in$ E to a positive integer, meaning: $\mu(e) \in \mathbb{N}$.
3. The number of colors used by an edge coloring $\mu$ is $|\{n \in \mathbb{N} : e \in E, \mu(e) = n\}|$, which we denote as the set $C$.
4. Two edges are called *adjacent* when they share a vertex, which can be written as: $e_1 \cap e_2 \neq \emptyset$, with $e_1, e_2 \in$ E.

**Translation to propositional logic**

As described above, for an edge coloring to be considered 'proper' there are a few things that need to be true about the color assignment. We will go through these conditions one by one and translate them to propositional logic. Afterwards we will demonstrate how these formulas help us solve the problem:

(I) The first thing that needs to hold for a color assignment is quite an 'obvious' condition, which is that every edge needs to be assigned a color. Meaning, if we have colors 1, 2 and 3, its not possible that an edge is assigned none of these colors. We translate this using the above defined input as follows:

$$\bigwedge_{e \in E} \left( \bigvee_{c \in C} \mu(e) = c \right)$$

This is a formal defintion but can be interpreted as follows: all the different edges we have in the graph ($e \in E$) need to be assigned at least one color $c \in C$. For example, when we have two edges $e_1$ and $e_2$ and three colors 1, 2 and 3, the formula written out will look as follows:

$$((\mu(e_1) = 1) \vee (\mu(e_1) = 2) \vee (\mu(e_1) = 3)) \wedge$$
$$((\mu(e_2) = 1) \vee (\mu(e_2) = 2) \vee (\mu(e_2) = 3))$$

Stating that $e_1$ needs to be assigned either color 1, 2 OR 3; AND that $e_2$ also needs to be assigned either color 1, 2 OR 3.

(II) The second condition that will need to hold follows from the first condition we defined. Similar to how every edge needs to be assigned at least a color, we can only assign a maximum of one color to an edge; we can not have an edge that is both blue and red for example. We need to define a propositional logic formula in CNF form that makes sure this holds, which is demonstrated below:

$$\bigwedge_{e \in E} \left( \bigwedge_{\{c_1, c_2\} \in C, c_1 \neq c_2} (\neg(\mu(e) = c_1) \vee \neg(\mu(e) = c_2)) \right)$$

Another quite formal definition, but it can be interpreted as: for every edge $e \in E$ (all the known edges) and for all the combinations of colors possible ($c_1$ and $c_2 \in C$) that are not the same ($c_1 \neq c_2$); it can not happen that two colors are both true; otherwise the whole expression will be false. Continuing on the example with two edges $e_1$ and $e_2$ and three colors 1, 2 and 3, the formula written out will look as follows:

$$(\neg(\mu(e_1) = 1) \vee \neg(\mu(e_1) = 2)) \wedge$$
$$(\neg(\mu(e_1) = 1) \vee \neg(\mu(e_1) = 3)) \wedge$$
$$(\neg(\mu(e_1) = 2) \vee \neg(\mu(e_1) = 3)) \wedge$$

$$(\neg(\mu(e_2) = 1) \vee \neg(\mu(e_2) = 2)) \wedge$$
$$(\neg(\mu(e_2) = 1) \vee \neg(\mu(e_2) = 3)) \wedge$$
$$(\neg(\mu(e_2) = 2) \vee \neg(\mu(e_2) = 3))$$

Here, the first three clauses make sure that for edge $e_1$ a maximum of one color is assigned, and the last three clauses make sure that the same is the case for edge $e_2$. Together with the rules defined in the first part, we will have exactly one color for each edge.

(III) The third, and arguably most important, constraint for a proper edge color assignment, is that no adjacent edges have the same color. As described above in the problem definition, two edges are called *adjacent* when they have a graph node or *vertex* in common: $e_1 \cap e_2 \neq \emptyset$. When translating this constraint to propositional logic, we get the following formula:

$$\bigwedge_{c \in C} \left( \bigwedge_{\{e_1, e_2\} \in E, e_1 \cap e_2 \neq \emptyset} (\neg(\mu(e_1) = c) \vee \neg(\mu(e_2) = c)) \right)$$

A third formal definition that might be difficult to intuitively understand immediately. It can be described as follows: for every color there is $(c \in C)$, all known adjacent edges in our graph (noted by $\{e_1, e_2\} \in E, e_1 \cap e_2 \neq \emptyset$) can not have the same color, which is enforced by the $\neg(\mu(e_1) = c) \vee \neg(\mu(e_2) = c)$. It states that at least one of the two adjacent edges needs to be not color $c$.

When we go back to our example of a graph that has two edges $e_1$ and $e_2$ and three colors 1, 2 and 3 and we specify that $e_1$ and $e_2$ are adjacent edges, we can write this formula out as follows:

$$\begin{gathered}
(\neg(\mu(e_1) = 1) \vee \neg(\mu(e_2) = 1)) \wedge \\
(\neg(\mu(e_1) = 2) \vee \neg(\mu(e_2) = 2)) \wedge \\
(\neg(\mu(e_1) = 3) \vee \neg(\mu(e_2) = 3))
\end{gathered}$$

Which shows that all the adjacent edges in our graph, they can never be both the same color.

To summarize we have formed three constraints that the edge coloring problem needs to satisfy: (I) every edge has at least one color, (2) every edge can have at most one color and (3) all adjacent edges can not have the same color. This can all be put together in a big CNF formula as follows, which is a conjuction of the found constraints:

$$\begin{aligned}
\varphi = \; & \bigwedge\nolimits_{e \in E} \left( \bigvee\nolimits_{c \in C} \mu(e) = c \right) \wedge \\
& \bigwedge\nolimits_{e \in E} \left( \bigwedge\nolimits_{\{c_1, c_2\} \in C, c_1 \neq c_2} (\neg(\mu(e) = c_1) \vee \neg(\mu(e) = c_2)) \right) \wedge \\
& \bigwedge\nolimits_{c \in C} \left( \bigwedge\nolimits_{\{e_1, e_2\} \in E, e_1 \cap e_2 \neq \emptyset} (\neg(\mu(e_1) = c) \vee \neg(\mu(e_2) = c)) \right)
\end{aligned}$$

To finish our example of a simple graph featuring two edges $e_1$ and $e_2$ and three colors 1, 2 and 3, with $e_1$ and $e_2$ being adjacent edges, the whole CNF formula would look like this (a conjunction of all the previously found formulas):

$$\begin{aligned}
\varphi = \; & ((\mu(e_1) = 1) \vee (\mu(e_1) = 2) \vee (\mu(e_1) = 3)) \wedge && \text{(clause 1)} \\
& ((\mu(e_2) = 1) \vee (\mu(e_2) = 2) \vee (\mu(e_2) = 3)) \wedge && \text{(clause 2)} \\
\\
& (\neg(\mu(e_1) = 1) \vee \neg(\mu(e_1) = 2)) \wedge && \text{(clause 3)} \\
& (\neg(\mu(e_1) = 1) \vee \neg(\mu(e_1) = 3)) \wedge && \text{(clause 4)} \\
& (\neg(\mu(e_1) = 2) \vee \neg(\mu(e_1) = 3)) \wedge && \text{(clause 5)} \\
& (\neg(\mu(e_2) = 1) \vee \neg(\mu(e_2) = 2)) \wedge && \text{(clause 6)} \\
& (\neg(\mu(e_2) = 1) \vee \neg(\mu(e_2) = 3)) \wedge && \text{(clause 7)} \\
& (\neg(\mu(e_2) = 2) \vee \neg(\mu(e_2) = 3)) \wedge && \text{(clause 8)} \\
\\
& (\neg(\mu(e_1) = 1) \vee \neg(\mu(e_2) = 1)) \wedge && \text{(clause 9)} \\
& (\neg(\mu(e_1) = 2) \vee \neg(\mu(e_2) = 2)) \wedge && \text{(clause 10)} \\
& (\neg(\mu(e_1) = 3) \vee \neg(\mu(e_2) = 3)) && \text{(clause 11)}
\end{aligned}$$

**Solving the problem**
We have now converted the problem to a problem in CNF normal form. The only thing left to consider now is how an SAT solver can help us find a proper edge coloring of an arbitrary input graph G that uses a *minimal* number of colors.

We have found a general CNF formula that, for an arbitrary graph G, can help us find which edges need to have which color. Putting this CNF formula into an SAT solver such as CDCL will give us a *satisfiable* or *not satisfiable* result.

When the formula is declared *not satisfiable* by the SAT solver it means that there are not enough colors declared and we will need to add an extra color to assign to certain edges.

The absolute minimum amount of colors possible for any arbitrary graph G is the amount of edges that the vertex with the *most* edges has. That is, if a certain vertex has 3 edges, at least 3 different colors are necessary. The maximum amount of colors is the total amount of verteces in the graph, since we can just assign a different color to each different edge in this scenario.

We can input the CNF formula using the minimum amount of colors possible into an SAT solver, which will result in either *satisfiability* or *non-satisfiability.* When the SAT solver finds that the formula is satisfiable we have found our minimum color solution. However, when the SAT solver finds the formula to be not satisfiable, we can add one extra color to the set of colors and run the SAT solver again. This process will repeat itself until we have found a satisfiable solution. As discussed earlier, a satisfiable solution will always be found, since assigning a different color to all edges is a (worst-case scenario) solution.

**Algorithm summary**

1. We translated the problem of edge-coloring to a CNF-formula $\varphi$ for an arbitrary graph $G$ and arbitrary set of colors $C$, based on three constraints that the edge coloring needs to satisfy.
2. The minimum amount of colors needed is the amount of edges that the vertex with the most edges has. The maximum amount of colors is the amount of edges in graph $G$.
3. Put the CNF formula into an SAT solver starting with the minimum amount of colors. Repeat this process, adding one color every time the CNF formula $\varphi$ is found to be unsatisfiable by the SAT solver. Do this until a satisfiable solution by the SAT solver is found.