An Introduction to SAT Solving

Applied Logic for Computer Science

UWO – December 3, 2017

Plan

Assignment

Definition

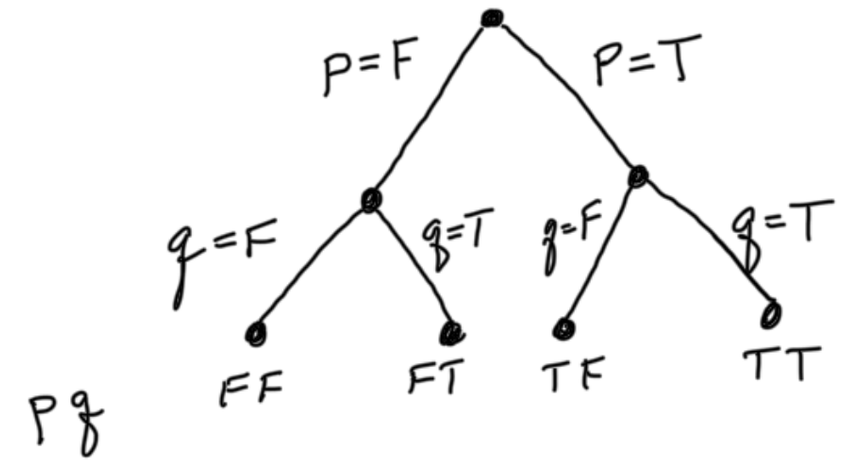
Let again V be a finite set of Boolean valued variables.

- An *assignment on* V is any map from V to $\{\text{false}, \text{true}\}$.
- Any assignment $\mathbf{v}$ on V induces an assignment on the propositional formulas on V by applying the following rules: if ϕ and ϕ' are propositional formulas on V then we have:
  1. $\mathbf{v}(\neg\phi) = \neg\mathbf{v}(\phi)$,
  2. $\mathbf{v}(\phi \wedge \phi') = \mathbf{v}(\phi) \wedge \mathbf{v}(\phi)'$,
  3. $\mathbf{v}(\phi \vee \phi') = \mathbf{v}(\phi) \vee \mathbf{v}(\phi')$,
  4. $\mathbf{v}(\phi \to \phi') = \mathbf{v}(\phi) \to \mathbf{v}(\phi')$,
  5. $\mathbf{v}(\phi \leftrightarrow \phi') = \mathbf{v}(\phi) \leftrightarrow \mathbf{v}(\phi')$.

Example

For the set of Boolean variables $V = \{p, q\}$. define $\mathbf{v}(p) = \text{false}$ and $\mathbf{v}(q) = \text{true}$. Then, we have:

- $\mathbf{v}(p \to q) = \text{true}$.
- $\mathbf{v}(p \leftrightarrow q) = \text{false}$.



Pg

The leaves of the tree represent the rows of the truth table
(i.e., complete assignments to the propositional variables)

Only one leaf node (i.e., row in the table; complete assignment)
needs to make the propositional formula true for
the formula to be satisfiable

So, let's generate the tree one leaf at a time

No because p=F

so $p \wedge (q \vee \neg p) \wedge (\neg q \vee \neg p)$ is F

F F T

and this is the answer

No because p=T and q=F

No because p=T and q=F

so, the 3rd conjunct is F

The formula is unsatisfiable.

Satisfiability (2/4)

Example

The formula is $p \wedge (q \vee \neg p) \wedge (\neg q \vee \neg p)$ is unsatisfiable. Indeed:

- the first clause, namely p, implies that p must be true,
- then, the second clause, namely $(q \vee \neg p)$, implies that q must be true,
- then, the third clause, namely $(\neg q \vee \neg p)$ is false,
- thus the whole formula cannot be satisfied.

Remarks

- Simple method for checking satisfiability: the *truth table* method, that is, check all 2^n possibilities for $\mathbf{v}$, where n is the number of variables in V.
- This always works, but has a running time growing exponentially with n.
- Practical algorithms and software solving SAT problems also run in time $O(2^n)$ but plays many tricks to terminate computations as early as possible.

SAT solvers

- Modern SAT solvers are based on *resolution*, and only apply to input formulas in *conjunctive normal form* (CNF).
- A *conjunctive normal* form (CNF) is a conjunction of clauses
- A *clause* is a disjunction of literals, say $a_1 \vee \cdots \vee a_n$, where $a_1, \ldots, a_n$ are literals.
- A *literal* is a Boolean variable or the negation of a Boolean variable
- Hence a CNF is of the form

$$\bigwedge_{1 \leq i \leq s} \left(\bigvee_{1 \leq j \leq t_j} \ell_{i,j} \right)$$

where the $\ell_{i,j}$'s are literals.

SAT solvers have many applications. Problems involving

- binary arithmetic
- program correctness
- termination of rewriting
- puzzles like Sudoku

can be encoded as SAT problems

Satisfiability (3/4)

Eight queens puzzle: general statement

- Go to `https://en.wikipedia.org/wiki/Eight_queens_puzzle` and read.
- For $n = 4$, there are two solutions.
- **Exercise**: how to phrase the search for those solutions into a SAT problem?
- **Hints**:
  - What should the Boolean variables represent?
  - What should the propositional formula represent?
- Remember the rules:
  - at most one (and at least one) queen in every row,
  - at most one (and at least one) queen in every column,
  - at most one queen in every diagonal.

Satisfiability (4/4)

Eight queens puzzle: case $n = 2$

- Associate a Boolean variable with each of the four corners, say a, b, c and d in clock-wise order.
- Exactly one queen on the top row writes: $(a \vee b) \wedge \neg(a \wedge b)$.
- Exactly one queen on the bottom row writes: $(c \vee d) \wedge \neg(c \wedge d)$.
- Exactly one queen on the left column writes: $(a \vee d) \wedge \neg(a \wedge d)$.
- Exactly one queen on the left column writes: $(b \vee c) \wedge \neg(b \wedge c)$.
- No two queens on the same diagonal writes: $\neg(a \wedge c) \wedge \neg(b \wedge d)$.
- We need some help to determine of the conjunction of these 5 formulas is satisfiable!

Plan

Satisfiability of a CNF formula

Notations

- Let $C_1, \ldots, C_m$ be clauses over a finite set V of Boolean variables $p_1, \ldots, p_n$.
- Define $\phi := C_1 \wedge \cdots \wedge C_m$.
- Let $\mathbf{v}$ be an assignment on V.

Remarks

Observations:

1. The CNF formula ϕ is satisfiable by $\mathbf{v}$ if, and only if, the clause C_i, for all $i \in \{1, \ldots, m\}$, is satisfiable by $\mathbf{v}$.
2. A clause C_i, for some $i \in \{1, \ldots, m\}$, is satisfiable by $\mathbf{v}$ if, and only if, at least one of its literals evaluates to true by $\mathbf{v}$.

Consequences:

- To check whether or not $\mathbf{v}$ satisfies ϕ, we may not need to know the truth values of all literals in all clauses.
- For instance, if $\mathbf{v}(p) =$ true and $\mathbf{v}(q) =$ false, we can see that the formula $\phi = (p \vee q \vee \neg r) \wedge (\neg q \vee s \vee q)$ is satisfied without considering $\mathbf{v}(r)$ and $\mathbf{v}(s)$.

Partial assignment

Definition

- A *partial assignment* on V is a partial function that associates to some Boolean variables of V a truth value (either true or false) and can be undefined for the others.
- Under a partial assignment on V, a clause C on V can be
  - *true* if one of its literals is true,
  - *false* (or *conflicting*) if all its literals are false,
  - *undefined* (or unresolved) if it is neither *true* not *false*.

Remarks

- Partial assignments allow us to construct assignments for a set of clauses incrementally, that is, one clause after another.
- The SAT solving algorithms to be presented next use partial assignments.
- They all start with an empty assignment (that is, the truth values of all Boolean variables are not defined) and try to extend this assignment, assigning one Boolean variable after another.

Partial assignments allow querying whether
the SAT problem has been solved

or
whether the branch will ultimately fail
without having to search all the way to
a leaf

Simplification of a formula by an evaluated literal (1/2)

Definition

For the CNF formula ϕ and a Boolean variables p, we denote by $\phi|_p$ the formula obtained from ϕ by replacing all occurrences of p by true and simplifying the result by removing:

- all true clauses,
- all false literals from undefined clauses.

The operation that maps (ϕ, p) to $\phi|_p$ is called the *simplification* of ϕ at p.
Similarly, we define $\phi|_{\neg p}$ as the formula obtained from ϕ by replacing all occurrences of p by false and simplifying the result by removing:

- all true clauses,
- all false literals from undefined clauses.

The operation that maps (ϕ, p) to $\phi|_{\neg p}$ is called the *simplification* of ϕ at $\neg p$.

Simplification of a formula by an evaluated literal (2/2)

Example

For $\phi := (p \vee q \vee \neg r) \wedge (\neg p \vee \neg r)$ we have:

$$\phi|_{\neg p} \quad \leftrightarrow \quad q \vee \neg r.$$

Proposition

For ϕ and p as in the above definition, we have:

- *if $\phi|_p$ is satisfiable, then ϕ is satisfiable too,*
- *if $\phi|_{\neg p}$ is satisfiable, then ϕ is satisfiable too.*
- *if ϕ is satisfiable then either $\phi|_p$ or $\phi|_{\neg p}$ is satisfiable.*

Remarks

- The above proposition is a key argument in the SAT solving algorithms to be presented hereafter.
- The proof of this proposition is easy and left as an exercise

Simplification can lead to two improvements

1. early discovery of success of failure
   (failure is more important because it occurs more often and reduces search)

2. can create unit clauses
   (see unit propagation later)

Principles

The SAT solving algorithms presented hereafter are based on the following ideas.

1. Each algorithm is stated as a recursive function taking a propositional formula ϕ and a partial assignment $\mathbf{v}$ as arguments.
2. These functions may modify their arguments:
   - the partial assignment can be extended,
   - the formula ϕ can be simplified at the newly assigned literal.
3. Before extending the assignment $\mathbf{v}$ and simplifying the formula ϕ, the function checks whether ϕ can be proved to be true or false, whatever are the values of the remaining unassigned Boolean variables.

Naive solver (1/3)

Input: A propositional formula ϕ on a finite set V and a partial assignment $\mathbf{v}$ on V such that only variables unassigned by $\mathbf{v}$ occur in ϕ.

Output: true if ϕ is satisfiable, false otherwise.

Naive_SAT$(\phi, \mathbf{v})$ $\{$

1. if every clause of ϕ has a true literal, return true;
2. if any clause of ϕ has all false literals, return false;
3. choose an $p \in V$ that is unassigned in $\mathbf{v}$;
4. assign p to true, that is, let $\mathbf{v}(p)$ = true;
5. if Naive_SAT$(\phi|_p, \mathbf{v})$ returns true, then return true; # this is a recursive call
6. assign p to false, that is, let $\mathbf{v}(p)$ = false;
7. if Naive_SAT$(\phi|_{\neg p}, \mathbf{v})$ returns true, then return true; # this is a recursive call
8. unassign $\mathbf{v}(p)$; # backtracking takes place here
9. return false; $\}$

Naive solver (2/3)

Remarks

The function call Naive_SAT$(\phi, \mathbf{v})$ can terminate early if:

- the formula is satisfied before all truth assignments are tested,
- all clauses are false before all variables have been assigned.

Proposition

The call Naive_SAT$(\phi, \mathbf{v})$ *terminates for all* ϕ *and* $\mathbf{v}$*.*

Naive solver (3/3)

Proposition

The call Naive_SAT$(\phi, \mathbf{v})$ *correctly decides whether* ϕ *is satisfiable.*

The pure literal rule

Proposition

Given a propositional CNF formula ϕ for which we check whether ϕ is satisfiable or not, if a variable is always positive *(that is, never appears with a $\neg$) or* always negative *(that is, always appears with a $\neg$) in ϕ, one only needs to set it to one value:* true *for positive variables,* false *for negative variables. Note that such variables are called* pure*.*

Proof

- Suppose p occurs only as positive literals in ϕ,
- If ϕ is satisfied by $\mathbf{v}$ and $\mathbf{v}(p) = \text{false}$, then ϕ is also satisfied by $\mathbf{v}'$ which is identical to $\mathbf{v}$ except that $\mathbf{v}'(p) = \text{true}$.
- so we do not need to try $\mathbf{v}(p) = \text{false}$.

Remarks

- Note that literals may become pure as variables are assigned.
- Indeed, as simplification happens, true clauses are removed.
- Hence, an unassigned variable which was not pure may become pure.

Unit propagation

Principle

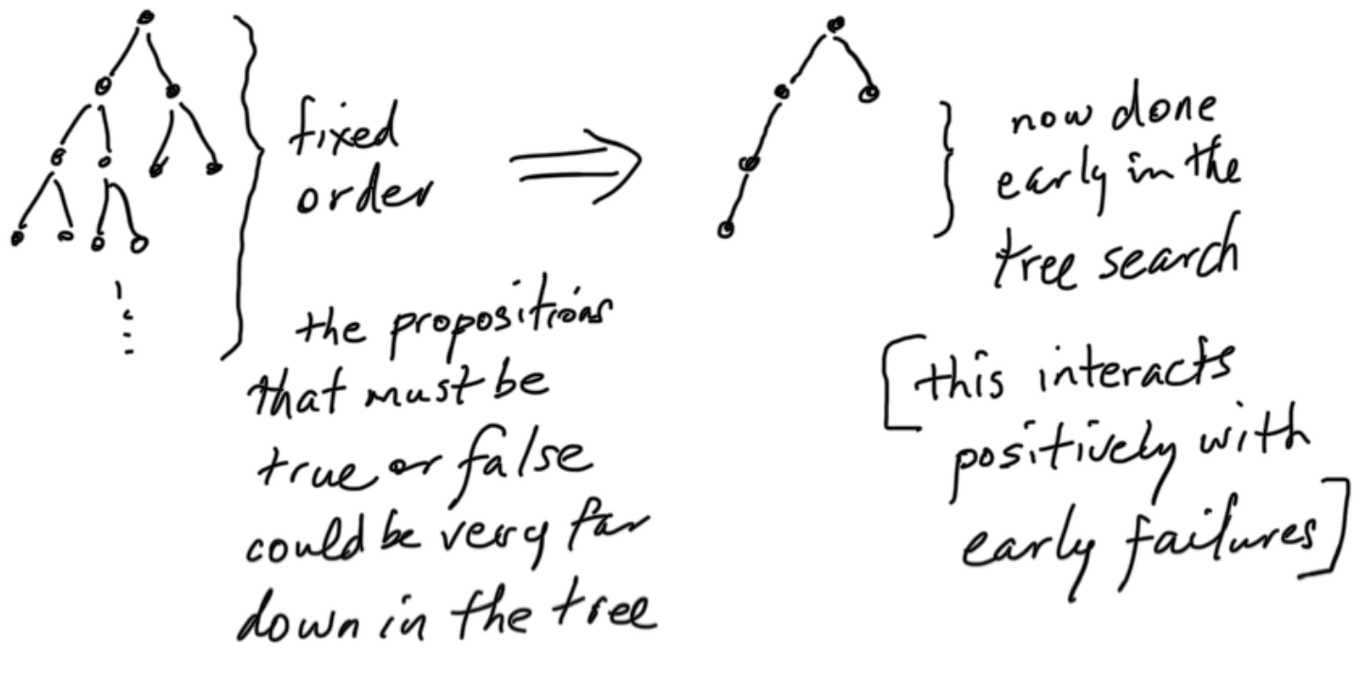
- Unit propagation (a.k.a Boolean constraint propagation or BCP) is a key component to fast SAT solving.
- Whenever all the literals in a clause are false except one, the remaining literal must be true in any satisfying assignment; such a clause is called a **unit clause**.
- Therefore, the algorithm can assign it to true immediately.
- After assigning a variable, there are often many unit clauses.
- Setting a literal in a unit clause often creates other unit clauses, leading to a cascade.
- Based on those observations, we define a sub-algorithm `BCP` as follows.
- The input specifications of BCP$(\phi, \mathbf{v})$ are the same as Naive_SAT$(\phi, \mathbf{v})$.
- Moreover, similarly to Naive_SAT$(\phi, \mathbf{v})$, the call BCP$(\phi, \mathbf{v})$may modify its arguments.

BCP$(\phi, \mathbf{v})$

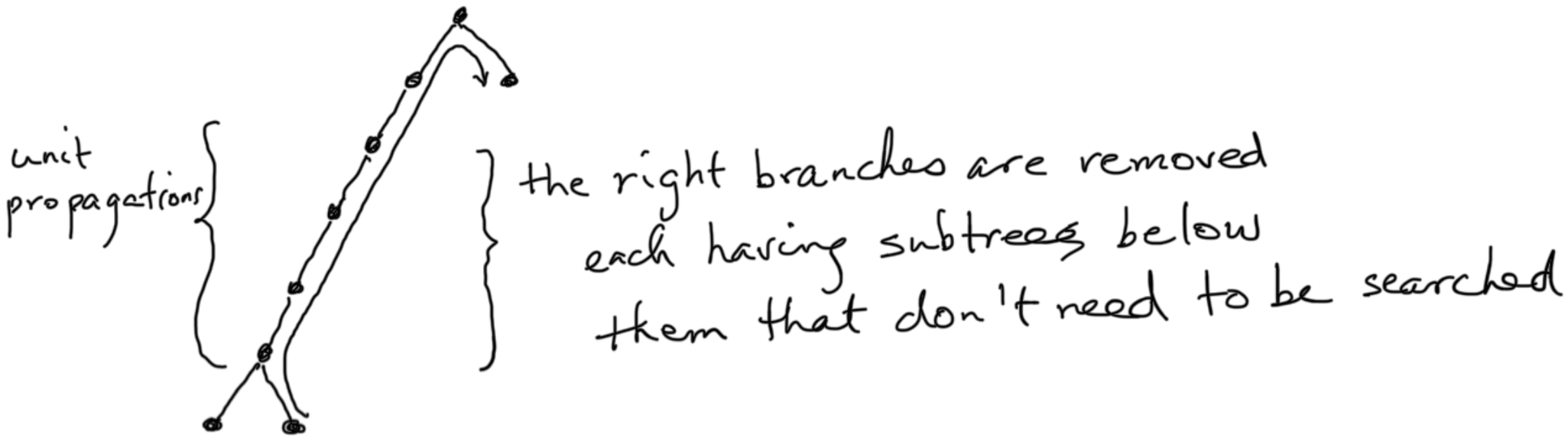
1. Repeatedly search for unit clauses, and set unassigned literal to required value.
2. If a literal is assigned to a conflicting value, then return false else return true.

Unit propagation achieves two improvements

1. Reorders the tree search



2. Subtrees are removed from search and backtracking can be more efficient



Davis-Putnam-Logemann-Loveland Algorithm (1/2)

Input: A propositional formula ϕ on a finite set V and a partial assignment $\mathbf{v}$ on V such that only variables unassigned by $\mathbf{v}$ occur in ϕ.

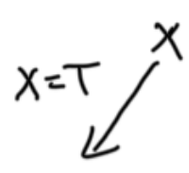
Output: true if ϕ is satisfiable, false otherwise.

DPLL$(\phi, \mathbf{v})$ {

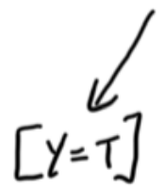
1. if every clause of ϕ has a true literal, return true;
2. if any clause of ϕ has all false literals, return false;
3. if BCP$(\phi, \mathbf{v})$ returns false, then return false;
4. choose an $p \in V$ that is unassigned in $\mathbf{v}$;
5. assign p to true, that is, let $\mathbf{v}(p)$ = true;
6. if DPLL$(\phi|_p, \mathbf{v})$ returns true, then return true;
7. assign p to false, that is, let $\mathbf{v}(p)$ = false;
8. if DPLL$(\phi|_{\neg p}, \mathbf{v})$ returns true, then return true;
9. unassign $\mathbf{v}(p)$; # backtracking takes place here
10. return false; }

Chronological Backtracking

$$\phi = \{ A \vee B,\ B \vee C,\ \neg A \vee \neg P \vee Q,\ \neg A \vee P \vee R,\ \neg A \vee \neg Q \vee R,\ \neg A \vee P \vee \neg R,\ \neg A \vee \neg Q \vee \neg R \}$$

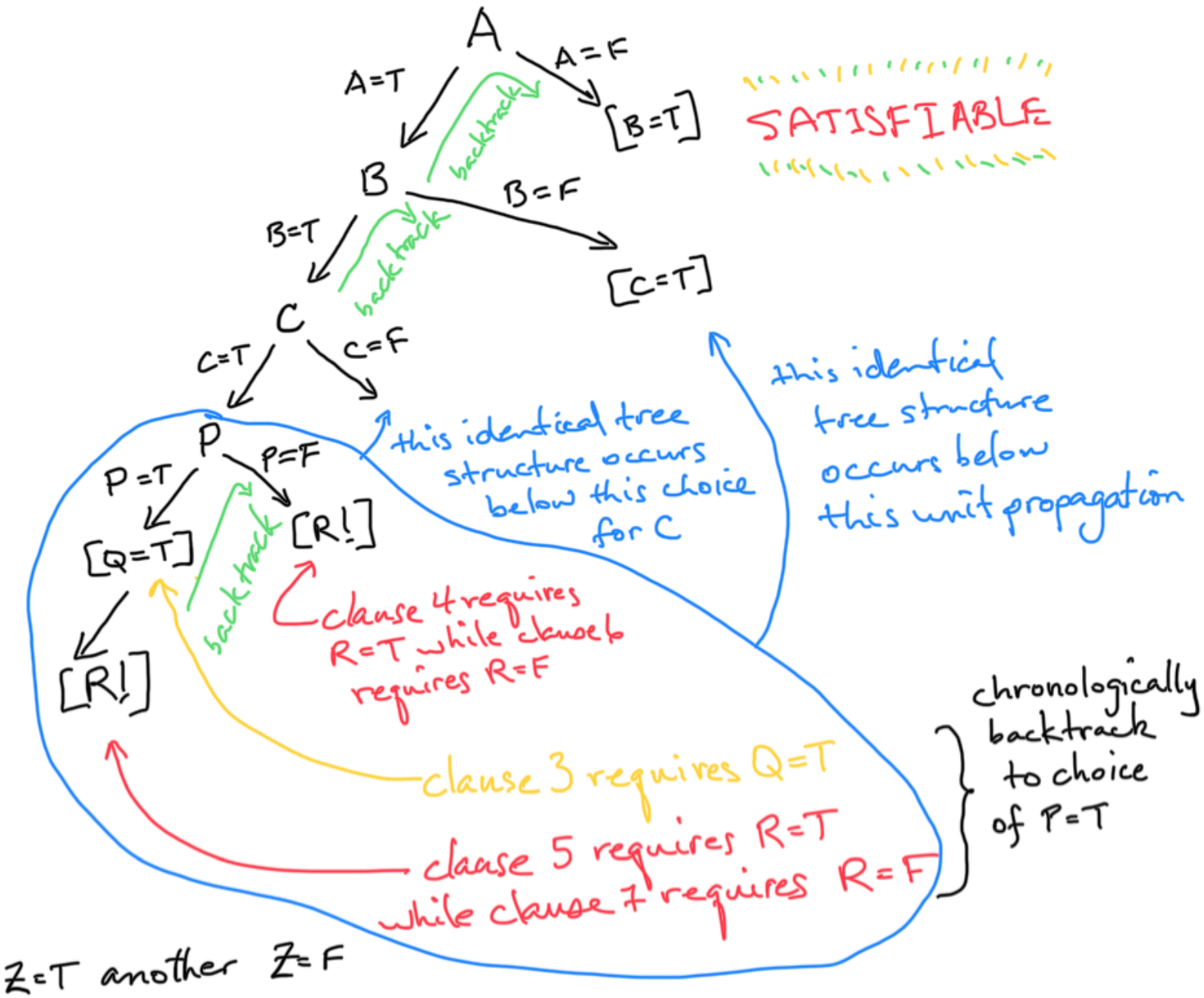


X is a decision node and the choice is X=T



Y is unit propagated and it must be T

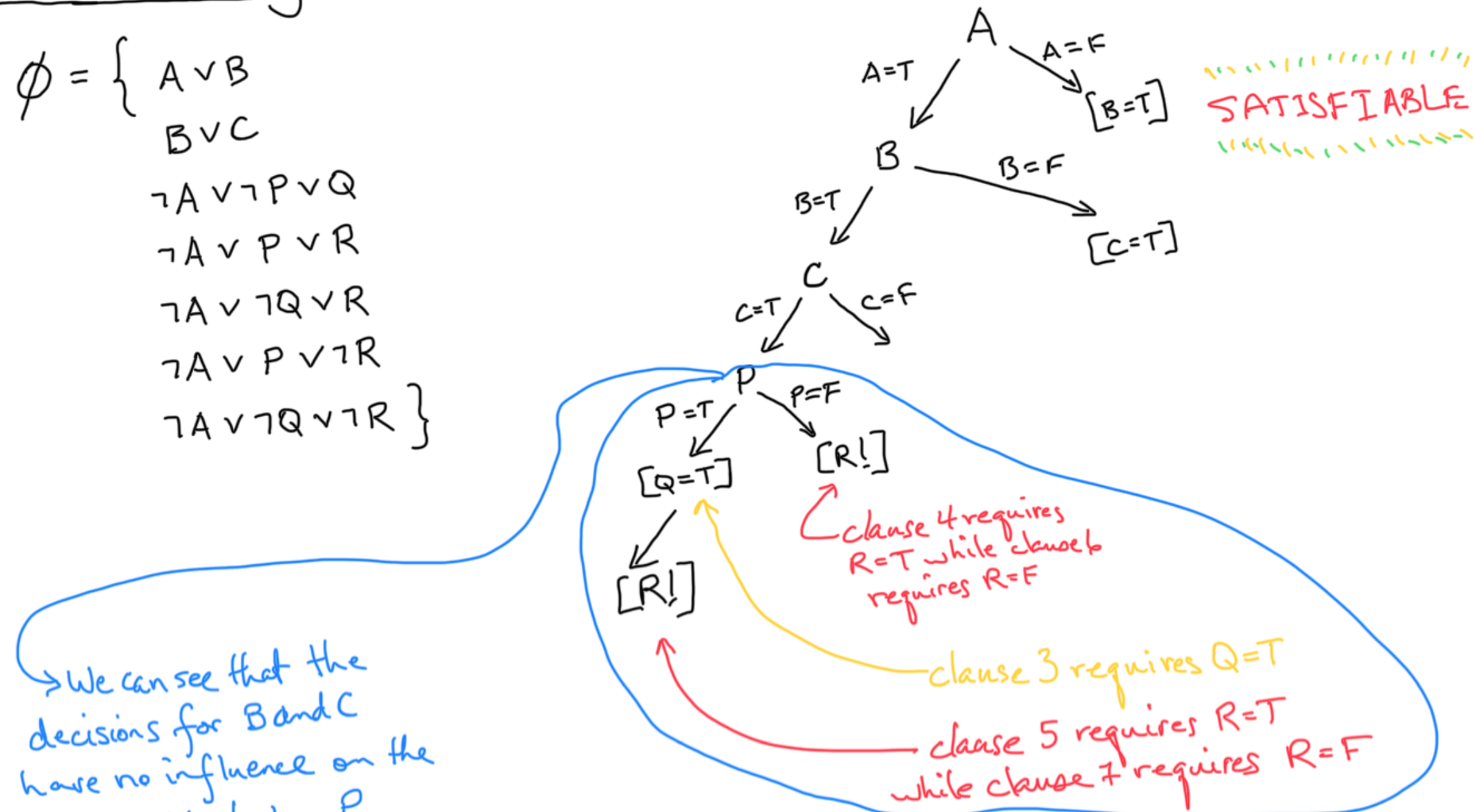
[Z !] there is a conflict
i.e., one clause would require Z=T another Z=F



SATISFIABLE

Non-Chronological Backtracking

$$\phi = \{ A \vee B,\ B \vee C,\ \neg A \vee \neg P \vee Q,\ \neg A \vee P \vee R,\ \neg A \vee \neg Q \vee R,\ \neg A \vee P \vee \neg R,\ \neg A \vee \neg Q \vee \neg R \}$$

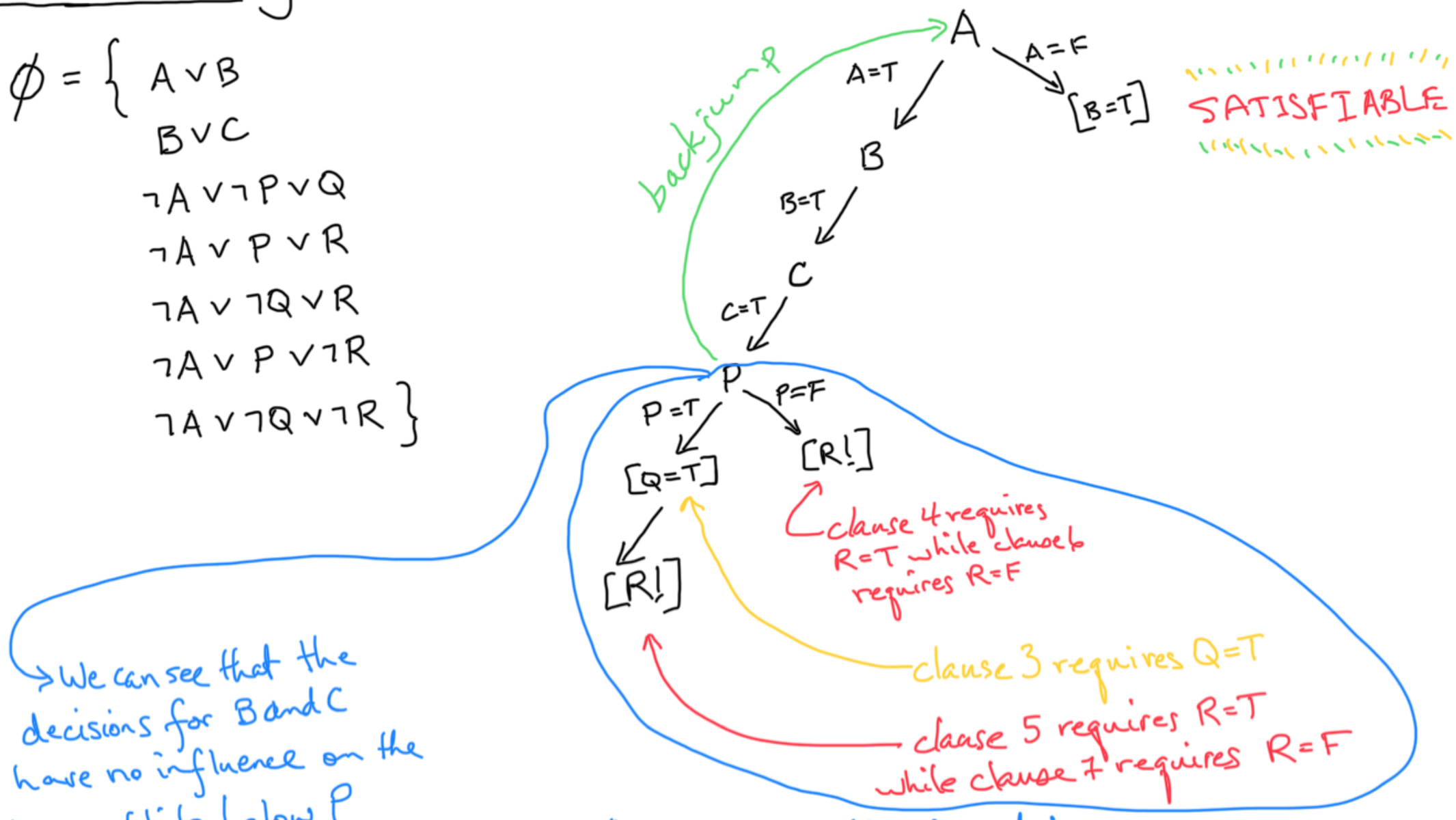


→ We can see that the decisions for B and C have no influence on the two conflicts below P

* we will see that an "Implication Graph" will give us this information

Non-Chronological Backtracking [a.k.a. Backjumping]

$$\phi = \{ A \vee B,\ B \vee C,\ \neg A \vee \neg P \vee Q,\ \neg A \vee P \vee R,\ \neg A \vee \neg Q \vee R,\ \neg A \vee P \vee \neg R,\ \neg A \vee \neg Q \vee \neg R \}$$



→ We can see that the decisions for B and C have no influence on the two conflicts below P

* we will see that an "Implication Graph" will give us this information

Implication Graph

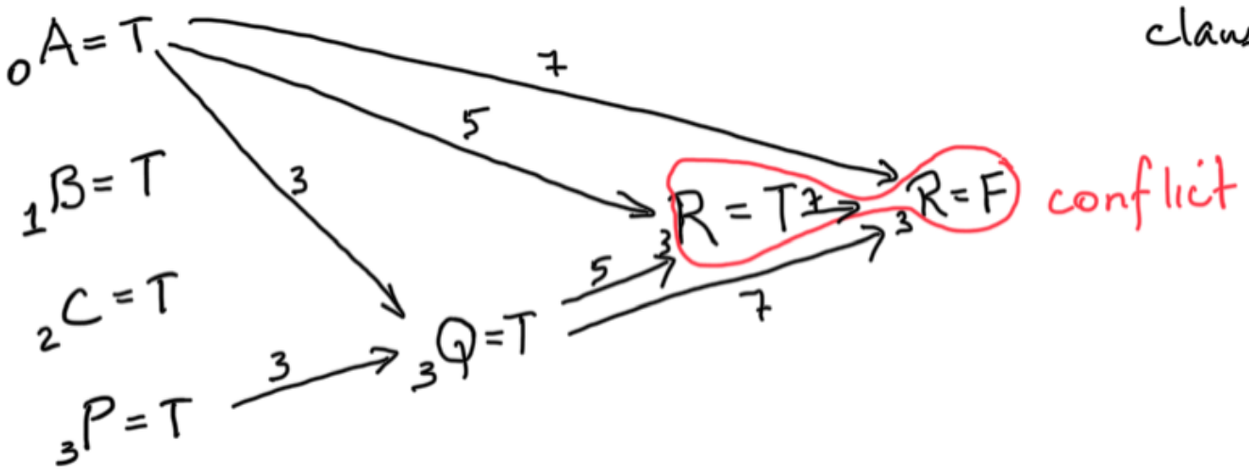
Some variables are decision variables (A, B, C, P in the previous example)

Some variables become unit variables (Q, R in the previous example)

When a unit variable is assigned a value, the implication graph indicates which variables contributed to that assignment

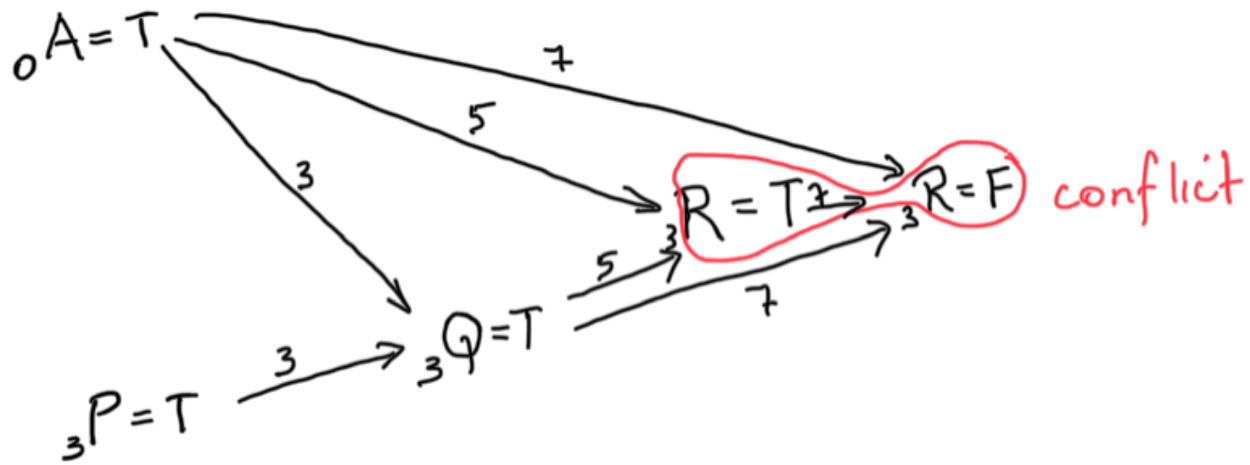
Each node in the graph indicates the assignment and the decision level.

We will label the edges with the clause that makes the unit assignment



Conflict Graph

Remove all nodes in implication graph
that do not connect to the conflict.



$$
\begin{aligned}
&(p_1 \vee p_4) \wedge \\
&(p_1 \vee \neg p_3 \vee \neg p_8) \wedge \\
&(p_1 \vee p_8 \vee p_{12}) \wedge \\
&(p_2 \vee p_{11}) \wedge \\
&(\neg p_7 \vee \neg p_3 \vee p_9) \wedge \\
&(\neg p_7 \vee p_8 \vee \neg p_9) \wedge \\
&(p_7 \vee p_8 \vee \neg p_{10}) \wedge \\
&(p_7 \vee p_{10} \vee \neg p_{12})
\end{aligned}
\quad = \quad \emptyset \qquad \text{(in CNF)}
$$

Is $\emptyset$ satisfiable?

$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

$p_2 \vee p_{11}$

$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

Step 1

Choose a variable to branch on

$p_1 = F$

$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

$p_2 \vee p_{11}$

$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

Step 2

Unit propagation

$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

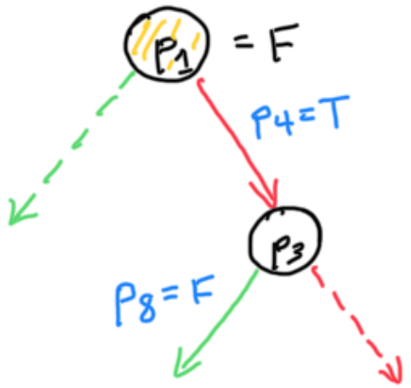
$p_2 \vee p_{11}$

$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

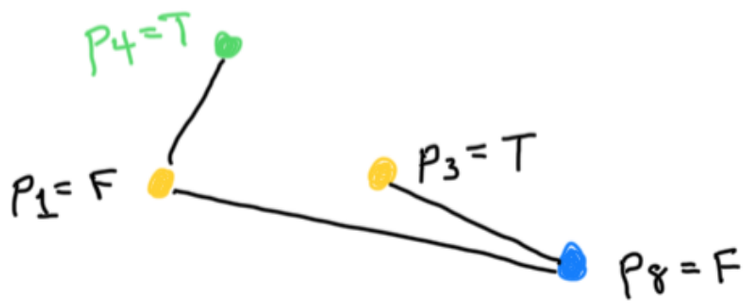


Step 3

Pick another branching variable (add to implication graph)

Step 4

Unit propagation and modify the implication graph



$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

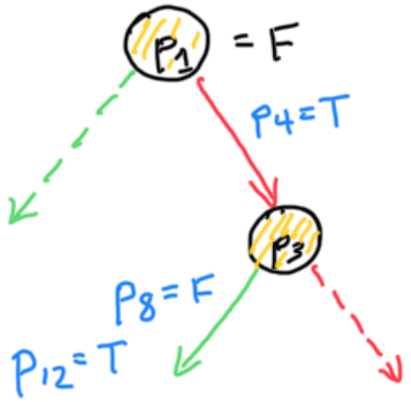
$p_2 \vee p_{11}$

$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

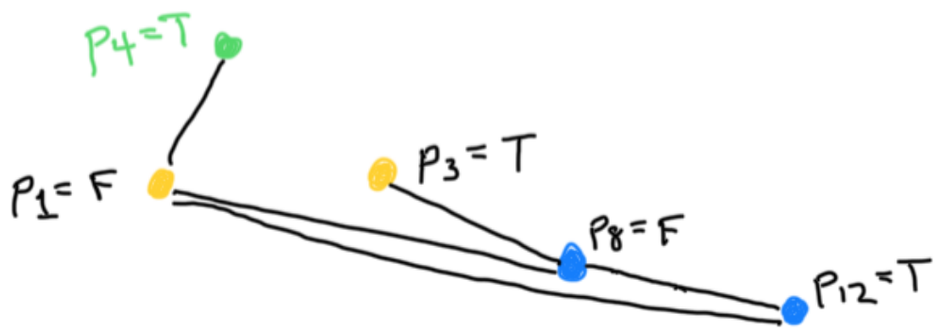


Step 3

Pick another branching variable (add to implication graph)

Step 4

Unit propagation and modify the implication graph



$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

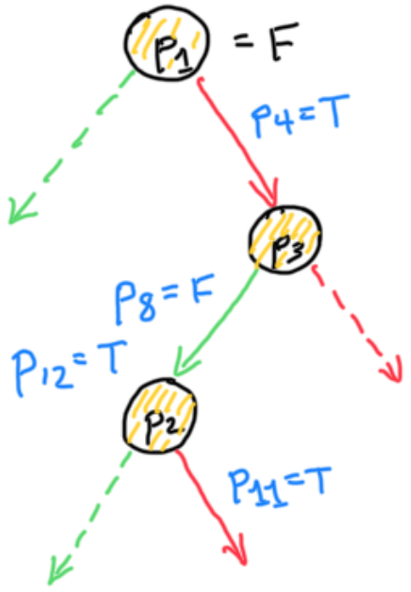
$p_2 \vee p_{11}$

$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

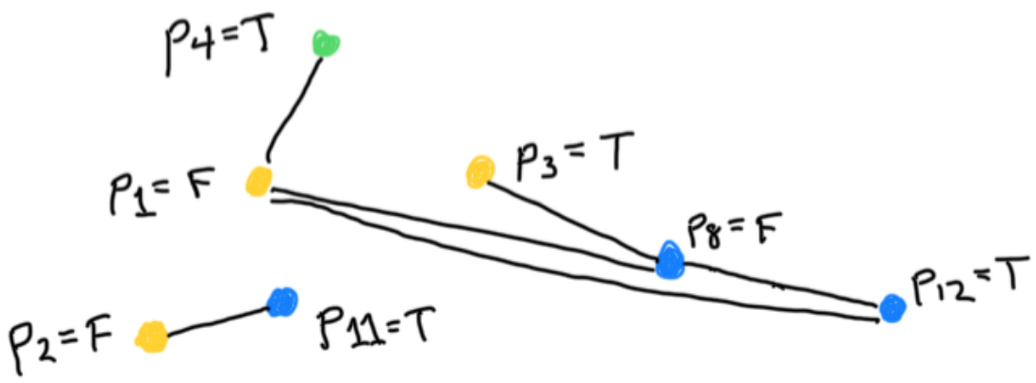


Step 5

Pick another branching variable (add to implication graph)

Step 6

Unit propagation and modify the implication graph



$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

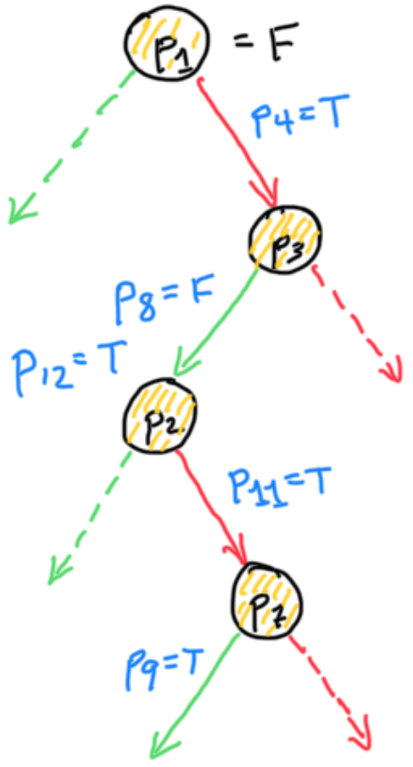
$p_2 \vee p_{11}$

$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

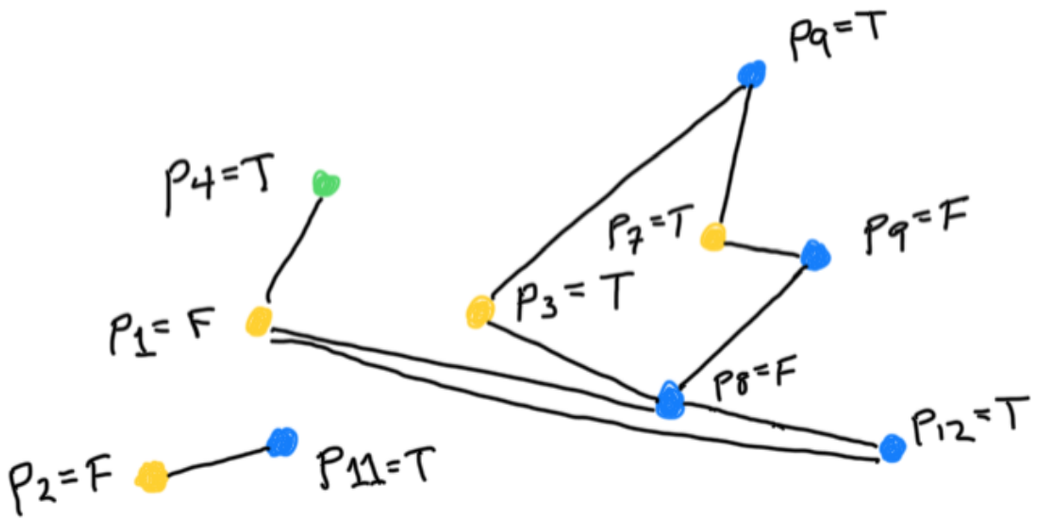


Step 7

Pick another branching variable (add to implication graph)

Step 8

Unit propagation and modify the implication graph



$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

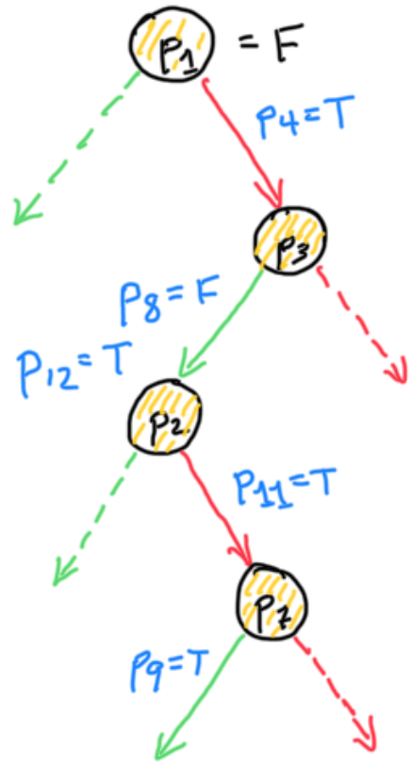
$p_2 \vee p_{11}$

$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$ ← clause is F!

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

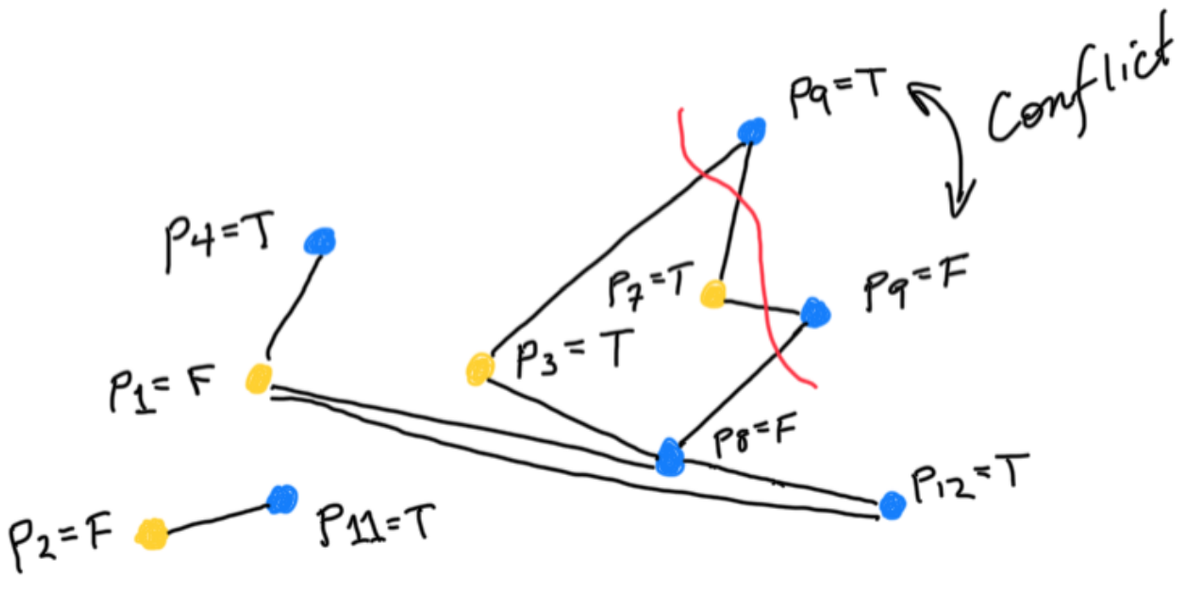


Step 9

Conflict found.
6th clause is F

Step 10

Locate the cut.
(Shown in red in implication graph.)
From the cut, find a conflicting condition.
(See next slide.)



Let's take the conflicting condition given by the implication graph
($p_3 = T$ will be written as the positive literal p_3, and
$p_8 = F$ will be written as the negative literal $\neg p_8$)

$$\phi \wedge p_3 \wedge p_7 \wedge \neg p_8 \rightarrow \bot$$

Take the contrapositive

$$\top \rightarrow \neg(\phi \wedge p_3 \wedge p_7 \wedge \neg p_8)$$

Since we have $\top$ implying the consequent, the consequent must be true. Change it to clause form

$$\neg\phi \vee \neg p_3 \vee \neg p_7 \vee p_8 \quad \equiv \quad \phi \rightarrow \underbrace{\neg p_3 \vee \neg p_7 \vee p_8}_{\text{"conflict clause"}}$$

Since the original problem ϕ implies the conflict clause, we can add this clause to ϕ without changing the satisfiability of ϕ

$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

$p_2 \vee p_{11}$

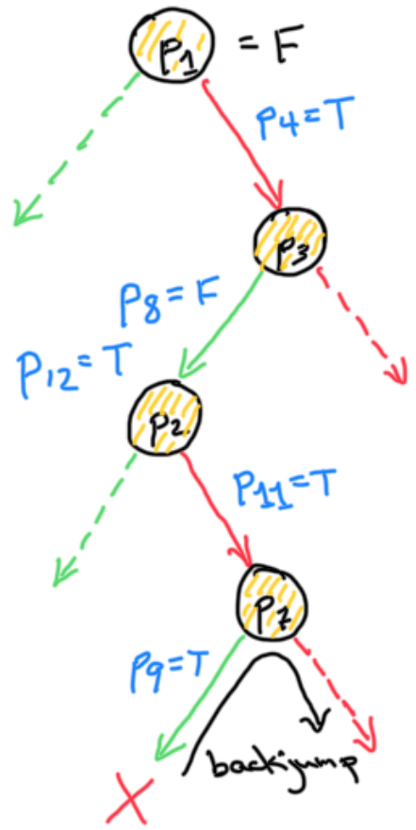
$\neg p_7 \vee \neg p_3 \vee p_9$

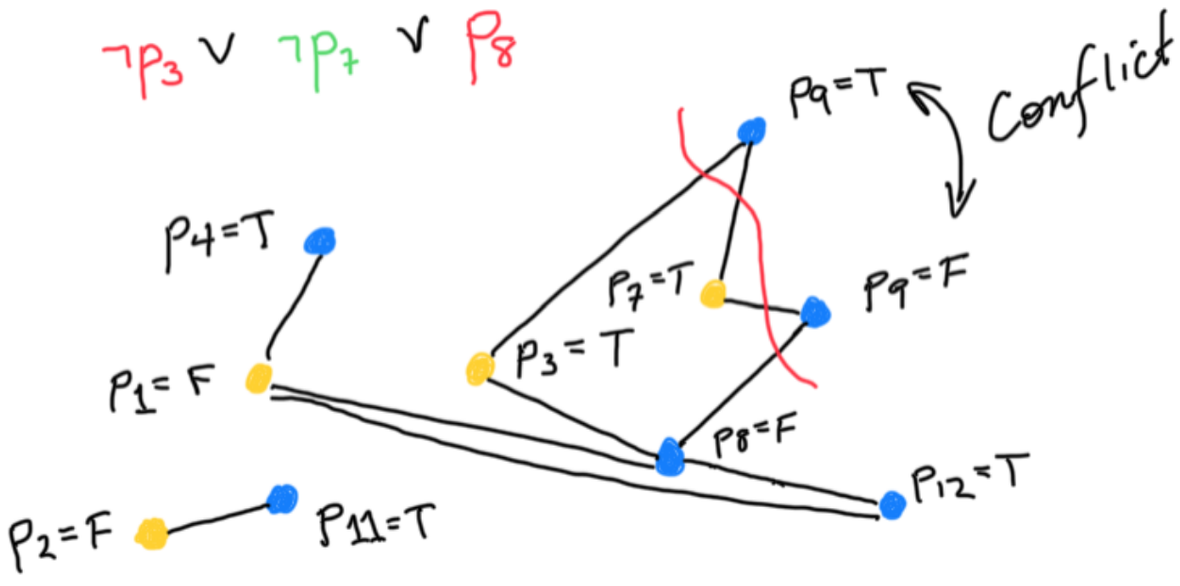
$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

$\neg p_3 \vee \neg p_7 \vee p_8$





Step 11

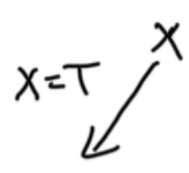
Make the negation of the conflicting condition a new clause

Step 12

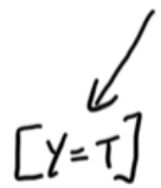
Add this learned clause to the set of clauses representing the problem and do a non-chronological back jump to the most recently assigned variable in the conflicting condition [in this case, it is p_7]

Chronological Backtracking

$$\phi = \{ A \vee B,\ B \vee C,\ \neg A \vee \neg P \vee Q,\ \neg A \vee P \vee R,\ \neg A \vee \neg Q \vee R,\ \neg A \vee P \vee \neg R,\ \neg A \vee \neg Q \vee \neg R \}$$

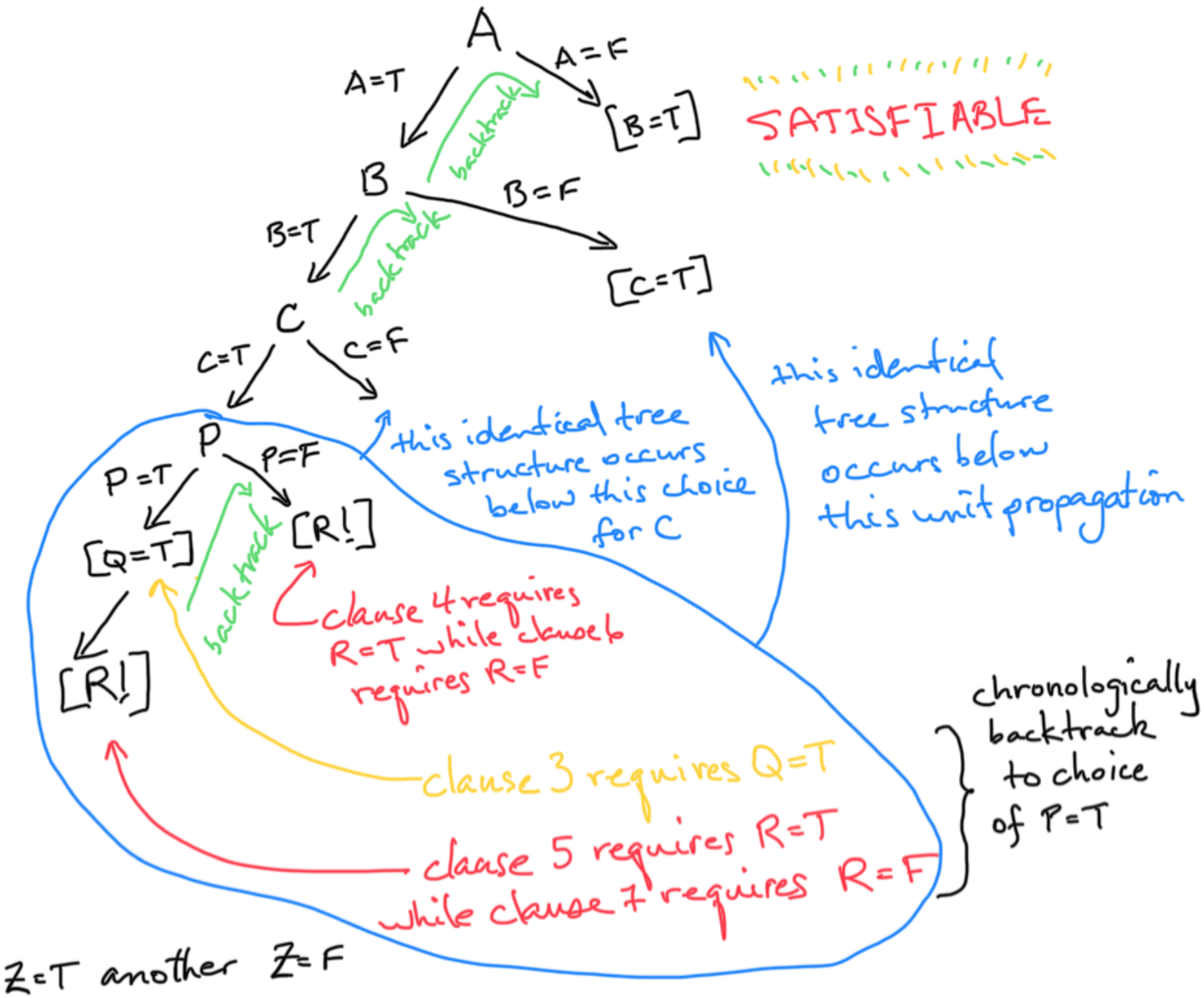


X is a decision node and the choice is X=T



Y is unit propagated and it must be T

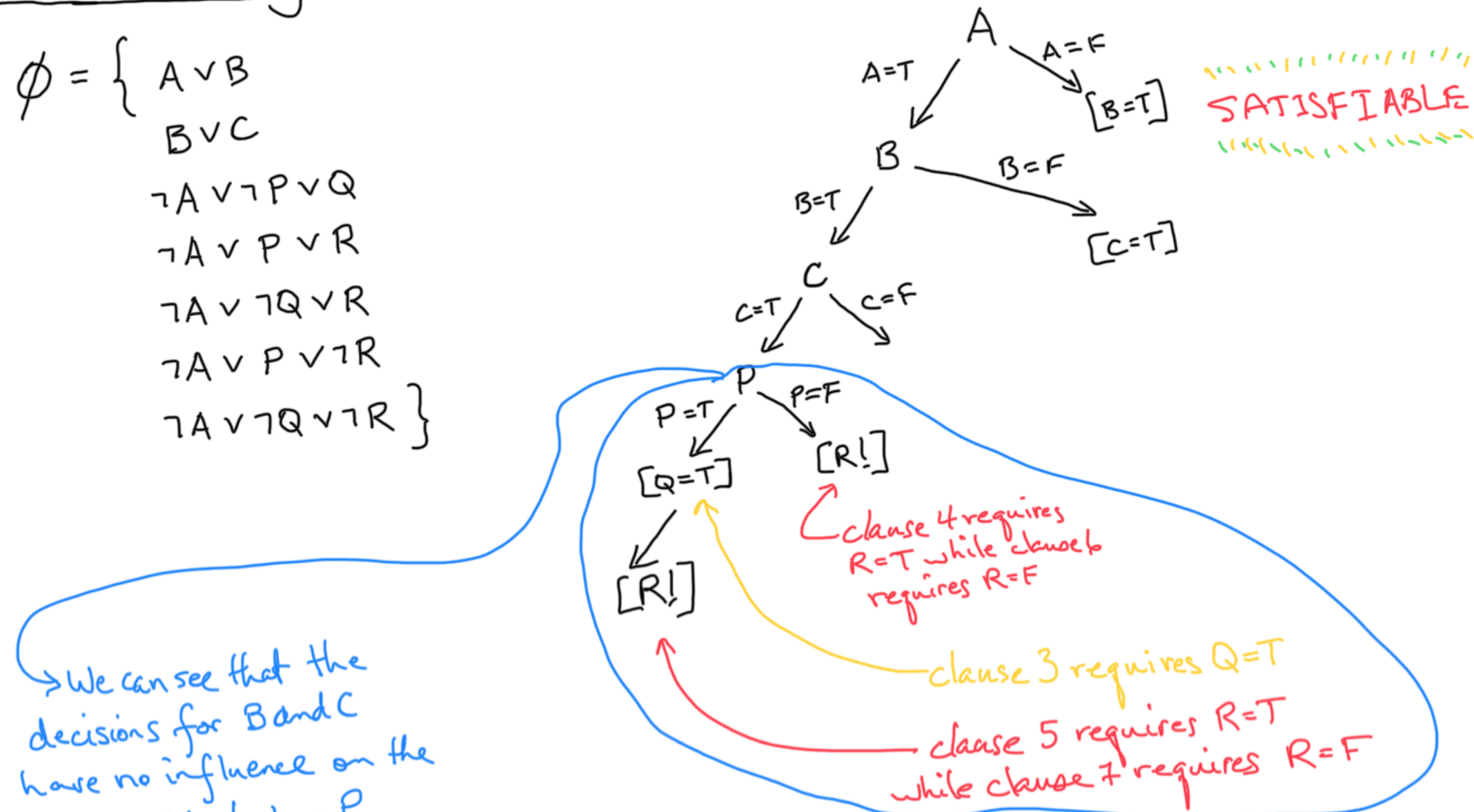
[Z !] there is a conflict
i.e., one clause would require Z=T another Z=F



SATISFIABLE

Non-Chronological Backtracking

$$\phi = \{ A \vee B,\ B \vee C,\ \neg A \vee \neg P \vee Q,\ \neg A \vee P \vee R,\ \neg A \vee \neg Q \vee R,\ \neg A \vee P \vee \neg R,\ \neg A \vee \neg Q \vee \neg R \}$$

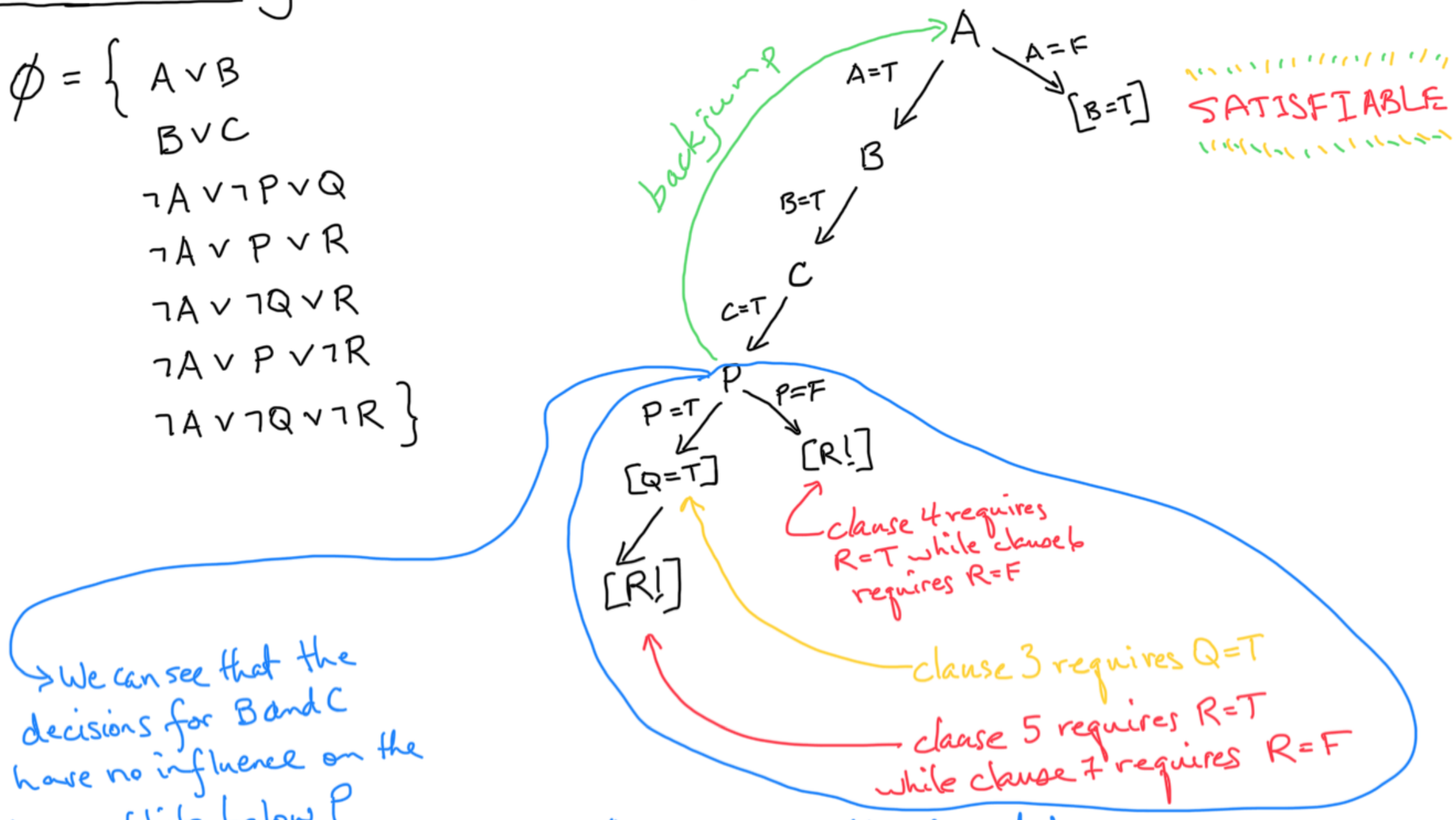


→ We can see that the decisions for B and C have no influence on the two conflicts below P

* we will see that an "Implication Graph" will give us this information

Non-Chronological Backtracking [a.k.a. Backjumping]

$$\phi = \{ A \vee B,\ B \vee C,\ \neg A \vee \neg P \vee Q,\ \neg A \vee P \vee R,\ \neg A \vee \neg Q \vee R,\ \neg A \vee P \vee \neg R,\ \neg A \vee \neg Q \vee \neg R \}$$



→ We can see that the decisions for B and C have no influence on the two conflicts below P

* we will see that an "Implication Graph" will give us this information

Implication Graph

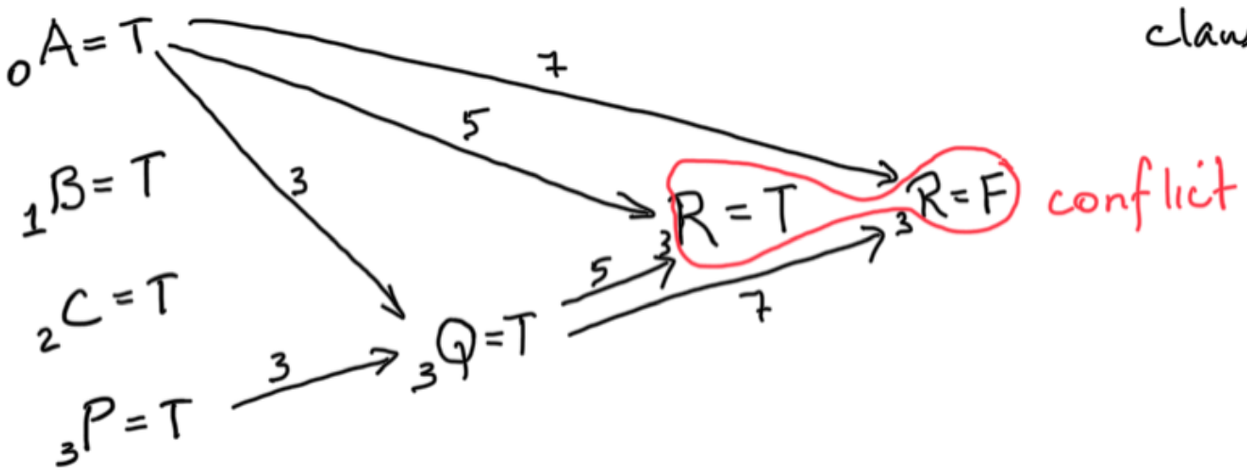
Some variables are decision variables (A, B, C, P in the previous example)

Some variables become unit variables (Q, R in the previous example)

When a unit variable is assigned a value, the implication graph indicates which variables contributed to that assignment

Each node in the graph indicates the assignment and the decision level.

We will label the edges with the clause that makes the unit assignment



$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

$p_2 \vee p_{11}$

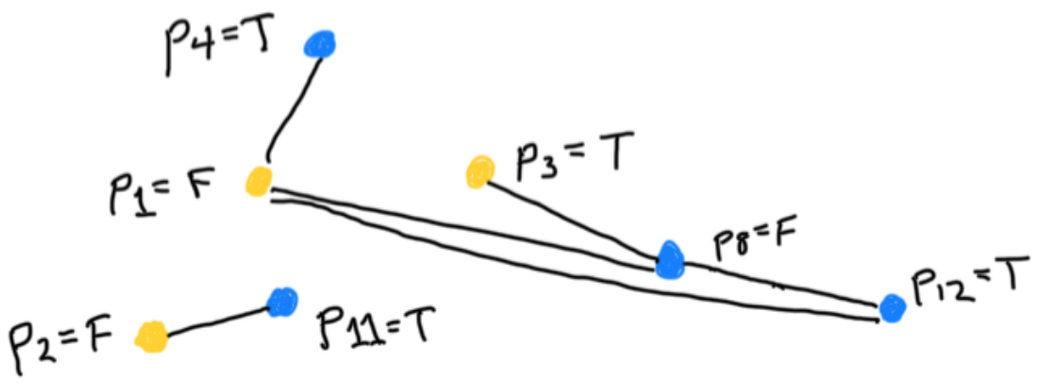
$\neg p_7 \vee \neg p_3 \vee p_9$

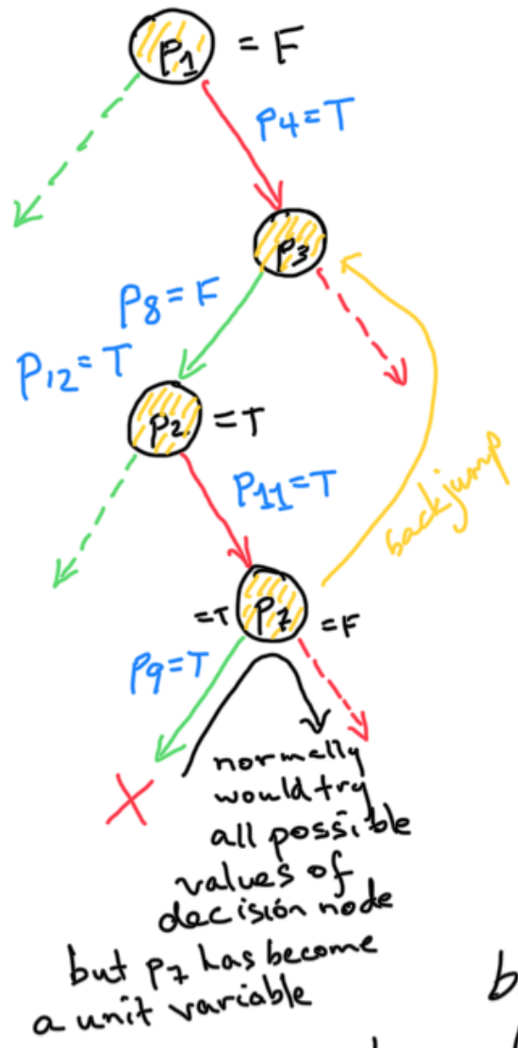
$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

$\neg p_3 \vee \neg p_7 \vee p_8$





Step 11

Make the negation of the conflicting condition a new clause

Step 12

Add this learned clause to the set of clauses representing the problem and do a non-chronological back jump to the most recently assigned variable in the conflicting condition [in this case, it is p_3]

$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

$p_2 \vee p_{11}$

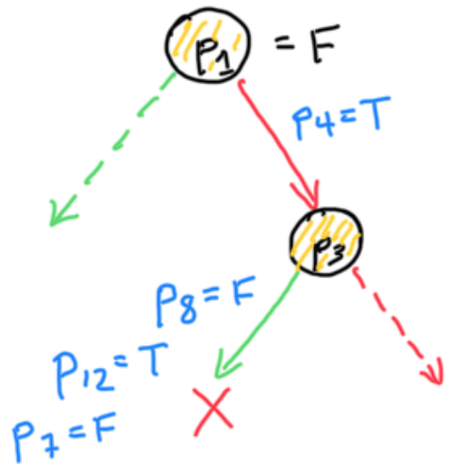
$\neg p_7 \vee \neg p_3 \vee p_9$

$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$ ← This clause is F!

$\neg p_3 \vee \neg p_7 \vee p_8$



$p_1 \vee \neg p_3$

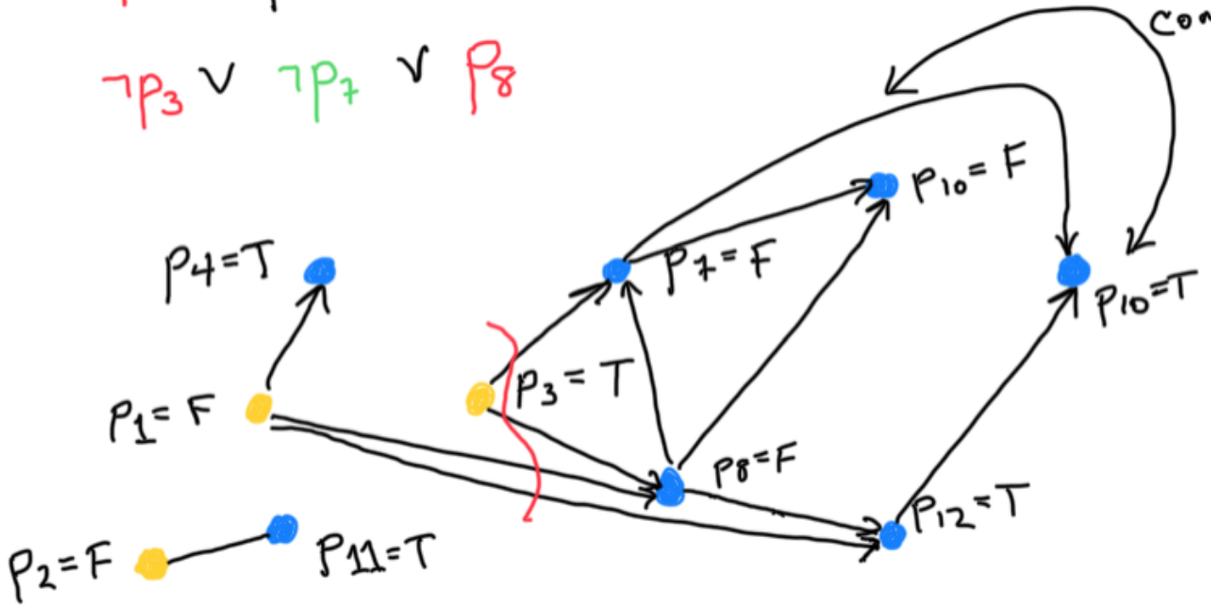
Step 13

Perform unit propagation at node p_3 and note another conflict

Step 14

Add this learned clause to the set of clauses representing the problem and do a non-chronological back jump to the most recently assigned variable in the conflicting condition

[in this case, it is p_1 because p_3 is now a unit variable]



$$\phi \wedge p_3 \wedge \neg p_1 \rightarrow \bot$$

$$\top \rightarrow \neg(\phi \wedge p_3 \wedge \neg p_1)$$

$$\neg\phi \vee \neg p_3 \vee p_1$$

$$\phi \rightarrow \neg p_3 \vee p_1$$

$p_1 \vee p_4$

$p_1 \vee \neg p_3 \vee \neg p_8$

$p_1 \vee p_8 \vee p_{12}$

$p_2 \vee p_{11}$

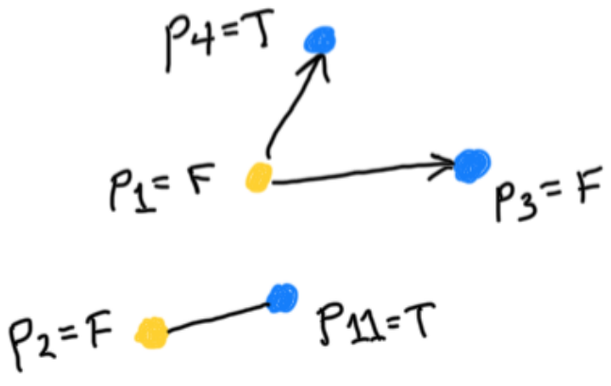
$\neg p_7 \vee \neg p_3 \vee p_9$

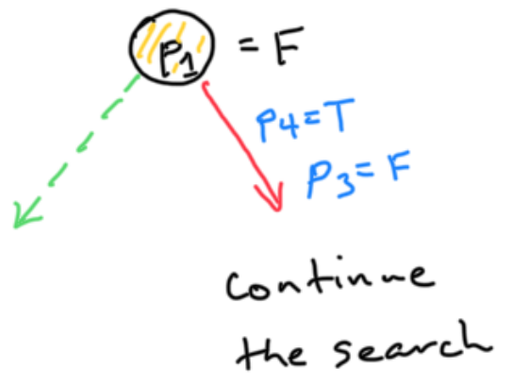
$\neg p_7 \vee p_8 \vee \neg p_9$

$p_7 \vee p_8 \vee \neg p_{10}$

$p_7 \vee p_{10} \vee \neg p_{12}$

$\neg p_3 \vee \neg p_7 \vee p_8$





$p_1 \vee \neg p_3$

Step 13

Perform unit propagation at node p_3 and note another conflict

Step 14

Add this learned clause to the set of clauses representing the problem and do a non-chronological back jump to the most recently assigned variable in the conflicting condition

[in this case, it is p_1 because p_3 is now a unit variable]