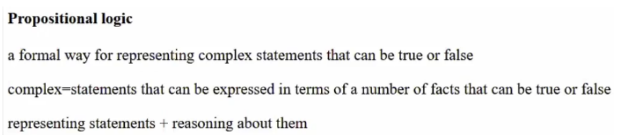
Planning and Reasoning

Sveva Pepe

Wednesday 25th November, 2020

1 Propositional Logic



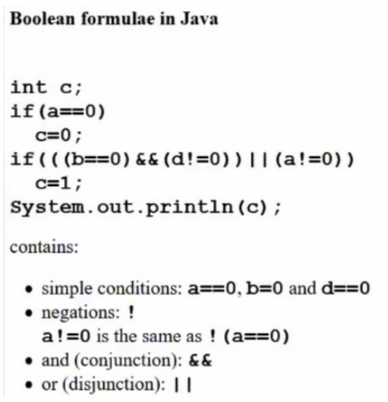
Propositional logic

a formal way for representing complex statements that can be true or false

complex=statements that can be expressed in terms of a number of facts that can be true or false

representing statements + reasoning about them

We have statements that can be true or false in which you have variables, boolean variables. Then we have combination like conjunction, disjuntion and negation.



Boolean formulae in Java

```
int c;
if(a==0)
  c=0;
if(((b==0)&&(d!=0))||(a!=0))
  c=1;
System.out.println(c);
```

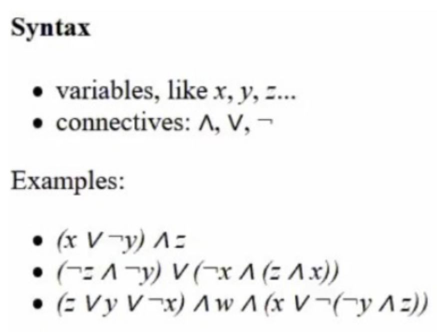
contains:

- simple conditions: `a==0`, `b=0` and `d==0`
- negations: `!`
  `a!=0` is the same as `!(a==0)`
- and (conjunction): `&&`
- or (disjunction): `||`

Example of program that combine boolean variables. In the first part of the course we have seen that planning program can be translate into SAT. So we can solve the problem just by establishing the satisfiability of propositional formula. Many other problems can be translate into SAT, any problem in NP can be translate into SAT, problem to establish propositional satisfiability.
You cannot encode problem of planning if you are lower exponentially non plan.

We can reduce our problem of satisfiability problem. Formula is satisfiable if there exist any satisfiable model. Usually we need model. Prove valididy means that formula should be satisfies by all model, we translate to satisfiability in which we can check if the negation of formula is satisfiable. If the negation is satisfiable our formula is not valid otherwise is valid. Fomula entails another is also translatable into validity and so into satisfiability.

1.1 Syntax



Syntax

- variables, like x, y, z...
- connectives: $\wedge, \vee, \neg$

Examples:

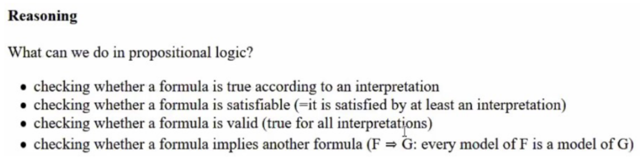
- $(x \vee \neg y) \wedge z$
- $(\neg z \wedge \neg y) \vee (\neg x \wedge (z \wedge x))$
- $(z \vee y \vee \neg x) \wedge w \wedge (x \vee \neg(\neg y \wedge z))$

Variables (x,y,z) are boolean, so can be only true or false. If we write variable x, this means that x==0.
Binary connective "and" ($\wedge$) and "or" $\vee$ and $\neg$.

1.2 Semantics

An *Interpretation* is evaluation of variables, so it tells the value of each variable.
When we say that an interpretation I is a **model** we mean that it is a complete assignement, for each variable there is a value.

1.3 Reasoning


Reasoning

What can we do in propositional logic?

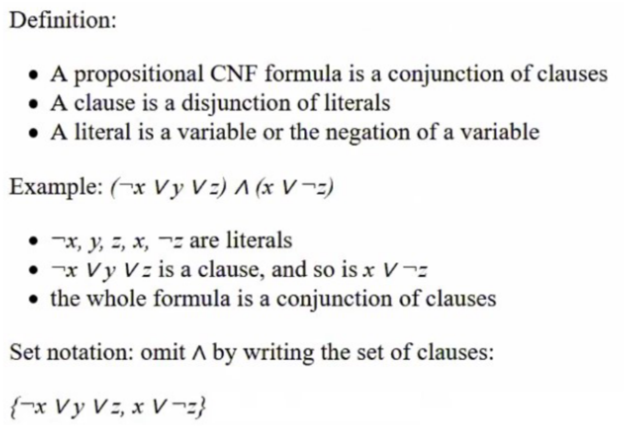
- checking whether a formula is true according to an interpretation
- checking whether a formula is satisfiable (=it is satisfied by at least an interpretation)
- checking whether a formula is valid (true for all interpretations)
- checking whether a formula implies another formula (F ⇒ G: every model of F is a model of G)

Formula is true according to interpretation? Given values of variables we just evaluate with $\wedge$ and $\vee$.
Check if the formula is satisfiable means that if there is an assignement that makes the formula true and this is complicated because, if we have for example 3 variables we have 8 possible evaluations (2^3). We need to combine them all, not just 6 values. For each value of x we have to combine all values of y and all values of z. The evaluation could be *exponential*.
This problem could occur also in case of validity, so whether the formula is true for all values.

The real problem is that we do not want to check every possible values because of the exponential number of them. The possible interpretations are exponentially many, so we do not want to check all of them, but we want to try to find only a few of them. Example of other problem that can be translated in propositional logic: *Planning, Scheduling* and *Diagnosis*.

1.4 CNF

An important subcase of propositional logic is the **CNF form**. Every formula can be translated into an equivalent CNF formula.


Definition:

- A propositional CNF formula is a conjunction of clauses
- A clause is a disjunction of literals
- A literal is a variable or the negation of a variable

Example: $(\neg x \vee y \vee z) \wedge (x \vee \neg z)$

- $\neg x$, y, z, x, $\neg z$ are literals
- $\neg x \vee y \vee z$ is a clause, and so is $x \vee \neg z$
- the whole formula is a conjunction of clauses

Set notation: omit $\wedge$ by writing the set of clauses:

$\{\neg x \vee y \vee z, x \vee \neg z\}$

CNF formula is a conjunction of clauses. Every clauses is a disjuntion of literals. Every literal is a variable of the negation of the variable.
In many cases is common to write a set of clauses without considering the conjunction symbol, $\wedge$.
Exmaples:

- x $\rightarrow$ one clause, made of single positive literal
- $x \wedge \neg y \rightarrow$ two clauses
- $(\neg z \vee y \vee w) \wedge (x \vee y) \wedge (\neg x \vee z \vee \neg w) \rightarrow$ three clauses

There are basically two ways to translate formula into CNF:

1. **Equivalent Conversion**

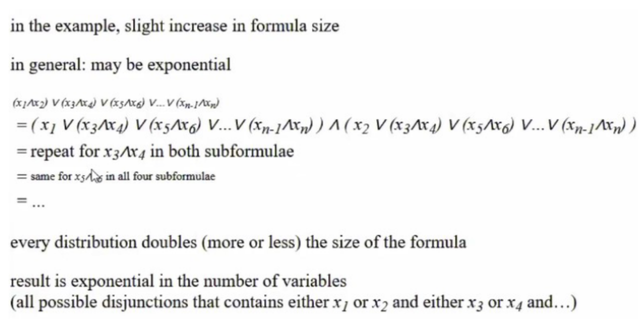
   The firt conversion method is basically a reformulation of the formula and it completely preserve the meaning of the formula.
   The idea is that in CNF formula we have negation applied only on literals. When we have negation of conjunction or disjuntion of formula the idea is to push negation inside, in such way we apply it only to literals. In other words we use the De Morgan's law.

$$\neg(A \wedge B) = \neg A \vee \neg B$$
$$\neg(A \vee B) = \neg A \wedge \neg B$$

   Example of this conversion:

$$\begin{aligned}\neg((x \wedge y) \wedge (z \vee \neg(\neg x \vee (z \wedge y)))) &= \neg(x \wedge y) \vee \neg(z \vee \neg(\neg x \vee (z \wedge y)))\\ &= (\neg x \vee \neg y) \vee (\neg z \wedge \neg\neg(\neg x \vee (z \wedge y)))\\ &= \neg x \vee \neg y \vee (\neg z \wedge (\neg x \vee (z \wedge y)))\\ &= (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg x \vee (z \wedge y))\\ &= (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee y)\end{aligned}$$

   The problem with this conversion is that it might exponential increase size.


in the example, slight increase in formula size

in general: may be exponential

$(x_1 \land x_2) \lor (x_3 \land x_4) \lor (x_5 \land x_6) \lor \ldots \lor (x_{n-1} \land x_n)$

$= (x_1 \lor (x_3 \land x_4) \lor (x_5 \land x_6) \lor \ldots \lor (x_{n-1} \land x_n)) \land (x_2 \lor (x_3 \land x_4) \lor (x_5 \land x_6) \lor \ldots \lor (x_{n-1} \land x_n))$

= repeat for $x_3 \land x_4$ in both subformulae

= same for $x_5 \land x_6$ in all four subformulae

= ...

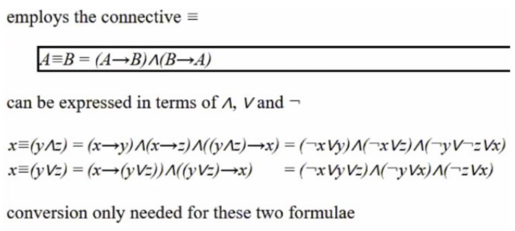
every distribution doubles (more or less) the size of the formula

result is exponential in the number of variables
(all possible disjunctions that contains either x_1 or x_2 and either x_3 or x_4 and…)

We can see from this example that every time we try to distribute one conjunction we have to double the rest of the formula.

2. **Equisatisfiable Conversion**

   The second formula is base on defined things, instead of just manipulationg the formula which may exponential increase size.


employs the connective ≡

$\boxed{A \equiv B = (A \rightarrow B) \land (B \rightarrow A)}$

can be expressed in terms of ∧, ∨ and ¬

$x \equiv (y \land z) = (x \rightarrow y) \land (x \rightarrow z) \land ((y \land z) \rightarrow x) = (\neg x \lor y) \land (\neg x \lor z) \land (\neg y \lor \neg z \lor x)$
$x \equiv (y \lor z) = (x \rightarrow (y \lor z)) \land ((y \lor z) \rightarrow x) = (\neg x \lor y \lor z) \land (\neg y \lor x) \land (\neg z \lor x)$

conversion only needed for these two formulae

We define subformulas, redefine formula in term of implication. We include new variables, so the formula is little different but still working, just looking at the satisfiability.

Example of this conversion:

$$
\begin{aligned}
&1)\neg((x \land y) \land (z \lor \neg(\neg x \lor (z \land y))))\\
&= \neg x \lor \neg y \lor (\neg z \land (\neg x \lor (z \land y)))\\
&2) x_1 \equiv \neg z \land (\neg x \lor (z \land y))\\
&\quad x_2 \equiv \neg x \lor (z \land y)\\
&\quad x_3 \equiv (z \land y)\\
&3)\neg x \lor \neg y \lor x_1\\
&\quad x_1 \equiv \neg z \land x_2\\
&\quad x_2 \equiv \neg x \lor x_3\\
&\quad x_3 \equiv (z \land y)
\end{aligned}
$$

First we push negation inside, this step does not increase the size. Then for all subformula we define variables and then we replace this variables inside.

The result might looks bigger but in fact it is only a **linear** increase size because for every "connective" we may add definition formula so the increase is only linear inside. Maybe we can double the size of the formula in the end, maybe could be linearly larger but not exponentially larger because in every step we replace subformula, with single variable, and then we have variable equivalent to something else which is also small. It is larger than the original but it is relative small.
The resulting formula is not exaclty equivalent to the original one but the point is that if the original formula is satisfiable then the new formula is satisfiable. Actually this transformation also preserve the number of models because the new formula is more or less the same of the original one but the only difference is presencce of new variables but the values of new variables are completely fixed one we know the values of the all variables. New variables are not too much a problem.

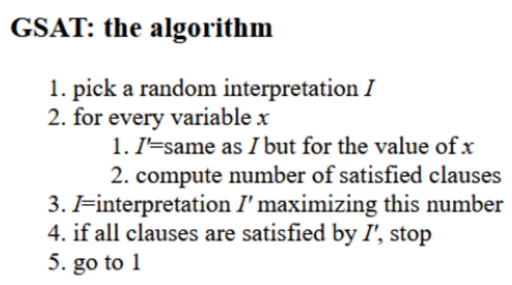
2 Two incomplete Methods for finding a model of a formula

Given a formula we want a model of the formula. We said the the formula is obtained by translating a planning problem. What we want is a plan and a plan is basically a model of this formula.
GSAT is extremely easy to implement. It starts from a random model and tries to modify that to satisfies as many clauses as possible.
Unit Propagation is based on a kind of inference, so if we know that a variable is true we propagate this information to other variables as well.

2.1 GSAT

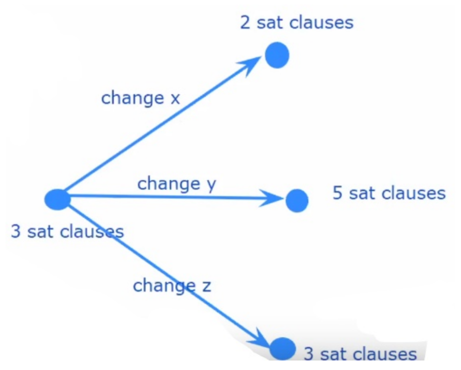
The idea is that we start with CNF and the main principle is that we have an aim: satisfied all clauses.
Let's say that we have a random interpretation I that satisfies 3 clauses of 7. This is quite far from our aim, but changing the values of the variables we increase this number. So we can have an improvement.
The main principle is that we start from a random interpretation and tries to change values of variables to increase the number of satisfied clauses until all clauses are satisfied. This is **not always possible**.

What we do?
The mechanism is simply that given a model we try to swap each individual variable (change the value of the variable) and if this can increase the number of satisfiable clauses we take it. We try this mechanism for all variables.



GSAT: the algorithm

1. pick a random interpretation I
2. for every variable x
    1. I'=same as I but for the value of x
    2. compute number of satisfied clauses
3. I=interpretation I' maximizing this number
4. if all clauses are satisfied by I', stop
5. go to 1



We start from random interpretation and we try to change value of a single variable. For example if you see the figure in the right we are in one step of the algorithm in which we have 3 variables to change and we can see that changing the variable y we are able to satisfied 5 clauses so we increase the number of satisfied clauses and we choose to change y. And then we restart from the new interpretation with this change on y. If we have only 5 clauses we have a model, otherwise we continue changing the values of variables at time.
It looks like very easy because we can think that at the end in at most n step, where n is the number of clauses, we should end but this is *not* the case because sometimes we end up with model that cannot be improved any further. The limit of this mechanism is that sometimes, just by moving one variable at time, we don't increase the number of satisfied clauses. But this number can be increased, so it is possible that we have a model that can be improved by changing more than 1 variable at the same times.
We end up in a condition in which we have **no local increase**, from the model there is no model that it is immediately accessible by changing one variable but there is a better one. We cannot find it because we have limited that we can change only one variable at time.
If we are stuck with a model that cannot be improved by local moves by changing one variable, we simply restart from a completely different model and hope for the best.

How it is incomplete?
The point is that if we end up with a model we are ok because the formula is satisfiable, otherwise we try to restart but in the end if after a number of possible attempts we don't find a model we don't know the result. This because maybe a model exists for that formula or the formula could be not satisfiable.

When is it incomplete?
There is no way to tell in advance if GSAT works for a particular formula or not. The only way we can do is just by run the algorithm and if we find a model ok otherwise we do not know.

2.2 Unit Propagation

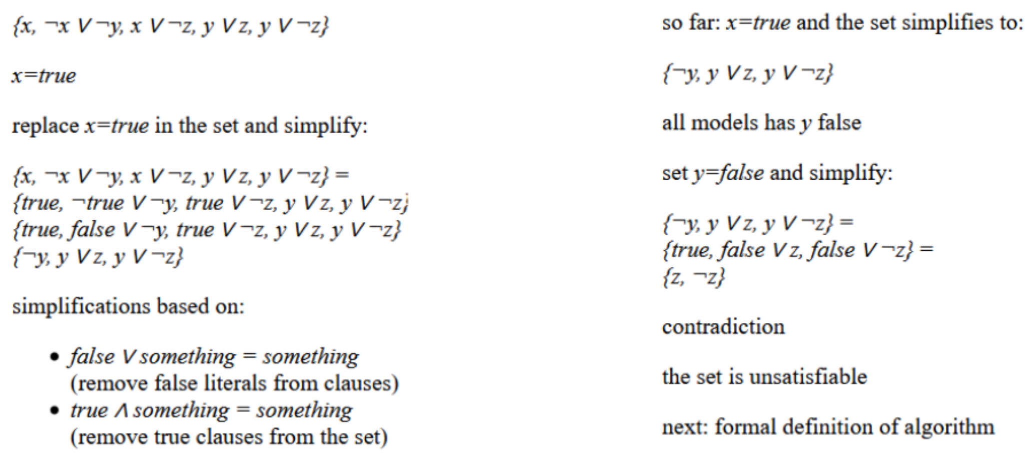
Unit Propagation is complete different method. It is still incomplete but it is incomplete in a completely different dimension because, first, we can tell in advance whether it will be incomplete or not; and second, if the unit propagation tells if the formula is unsatisfiable then the formula is unsatisfiable otherwise we do not know.
The unit propagation has 3 possible outcomes: satisfiable, unsatisfiable and unknown.
Another difference is that there is a particular subclass of formula in which Unit Propagation is known to be complete.
The idea is that if we have a clause made by a single literal, we have no choice because since we want to satisfied all clauses we need to satisfy also these particular clauses (with single literal). The only way we can satisfies clauses that only contain that particular literal is to make the literal true. This is important because it has an

effect on the other variables.
Suppose to have:

$$\{x, \neg x \vee y, x \vee \neg z, y \vee z, y \vee \neg z\}$$

We want to find a model in which all these clauses should be true.
Immediately the only way x must be true in the model is that the model makes x true, so the model has to contain $\{x = true\}$. It is the only way the model could satisfies all clauses.
Then the only way to make the second clause true is to make $\{y = true\}$. For the third clause we know that x is true so this clause is already true, so we can remove this clause it is consider redundant.
This mechanism is called Unit Propagation because we start with unit clauses[1], we set the value of the variable/variables and then we propagate, so we set the value of the variables in the other clauses.

The full example is:



$\{x, \neg x \vee \neg y, x \vee \neg z, y \vee z, y \vee \neg z\}$

x=*true*

replace x=*true* in the set and simplify:

$\{x, \neg x \vee \neg y, x \vee \neg z, y \vee z, y \vee \neg z\}$ =
$\{true, \neg true \vee \neg y, true \vee \neg z, y \vee z, y \vee \neg z\}$
$\{true, false \vee \neg y, true \vee \neg z, y \vee z, y \vee \neg z\}$
$\{\neg y, y \vee z, y \vee \neg z\}$

simplifications based on:

- *false* $\vee$ *something* = *something*
  (remove false literals from clauses)
- *true* $\wedge$ *something* = *something*
  (remove true clauses from the set)

so far: x=*true* and the set simplifies to:

$\{\neg y, y \vee z, y \vee \neg z\}$

all models has y false

set y=*false* and simplify:

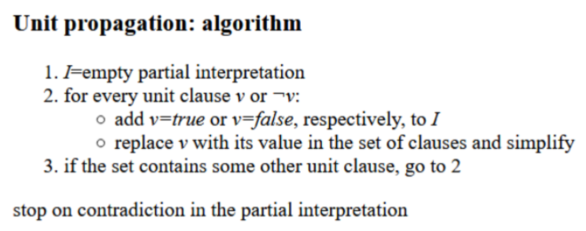
$\{\neg y, y \vee z, y \vee \neg z\}$ =
$\{true, false \vee z, false \vee \neg z\}$ =
$\{z, \neg z\}$

contradiction

the set is unsatisfiable

next: formal definition of algorithm

We consider first $\{x = true\}$ and then do the replacement into formulas and make semplification. Semplification rules are *false* $\vee$ *something* = *something* and *true* $\wedge$ *something* = *something*. Now on what I get from these semplification is a new formula in which I have a new unit clase, $\neg y$ (see the right side) so I do the same. In all model we should have that $\{y = false\}$. We propagate this information and do simplification but we find a contraddiction because the formula contains $\{z, \neg z\}$. This means that the formula is unsatisfiable.
The point is that we have to store the assignement somewhere, so what I did is to actually have these partial interpretations that accumulate these partial values. Initially a partial interpretation is an assignement to some of the variables, in particular, at the beginning it is empty. Then everytime I find a unit clause I set the value of the variable in the interpretation and proceed in this way. At the end there are 3 possibilities: *get contraddiction* (**unsatisfiable**), *all clauses are satisfied* (**satisfiable**) and *I stop at some point because no unit clause is left* (**unknown**).

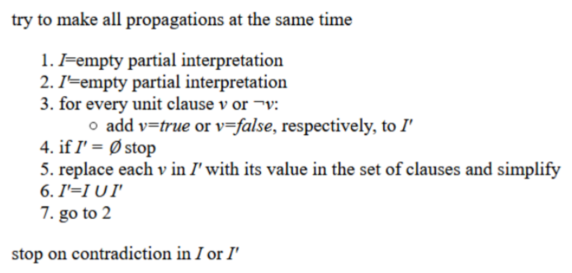


Unit propagation: algorithm

```
1. I=empty partial interpretation
2. for every unit clause v or ¬v:
      ○ add v=true or v=false, respectively, to I
      ○ replace v with its value in the set of clauses and simplify
3. if the set contains some other unit clause, go to 2

stop on contradiction in the partial interpretation
```

We start with empty partial interpretation. For every unit clause I set the value of the variable in the partial interpretation, then replace the value in the set of clauses and simplify. I continue until in every step I have an unit clause.



try to make all propagations at the same time

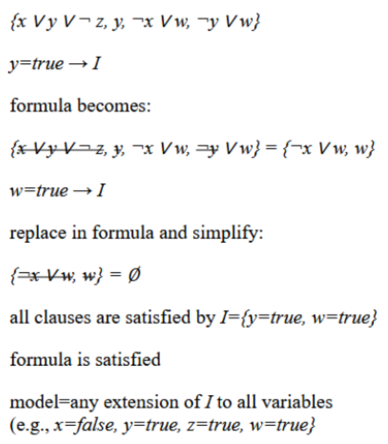
```
1. I=empty partial interpretation
2. I'=empty partial interpretation
3. for every unit clause v or ¬v:
      ○ add v=true or v=false, respectively, to I'
4. if I' = ∅ stop
5. replace each v in I' with its value in the set of clauses and simplify
6. I'=I ∪ I'
7. go to 2

stop on contradiction in I or I'
```

In practise, I try to not do the propagation one at time but try all of them. Scan the formula, collect all unit clauses in the partial interpretation and then try to simplify all of them at the same time. Then proceed.

[1]clause that contains only one literal

Unit propagation of consistent formula



{x ∨ y ∨ ¬ z, y, ¬x ∨ w, ¬y ∨ w}

y=true → I

formula becomes:

{~~x ∨ y ∨ ¬ z~~, ~~y~~, ¬x ∨ w, ~~¬y~~ ∨ w} = {¬x ∨ w, w}

w=true → I

replace in formula and simplify:

{~~¬x ∨ w~~, ~~w~~} = ∅

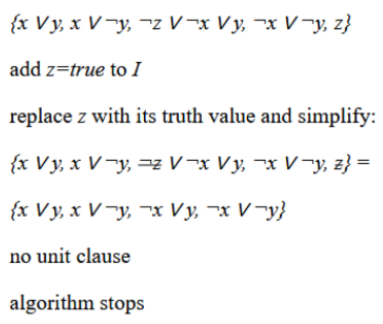
all clauses are satisfied by I={y=true, w=true}

formula is satisfied

model=any extension of I to all variables
(e.g., x=false, y=true, z=true, w=true}

At first $\{y = true\}$. Formula that contains y is simplified and also $\neg y$ is deleted because $\{y = true\}$. Then I set $\{w = true\}$, then replace and simplified. We end with empty set so all clauses are satisfied, the model is $I = \{y = true, w = true\}$.

Unit propagation of inconsistent formula



{x ∨ y, x ∨ ¬y, ¬z ∨ ¬x ∨ y, ¬x ∨ ¬y, z}

add z=true to I

replace z with its truth value and simplify:

{x ∨ y, x ∨ ¬y, ~~¬z~~ ∨ ¬x ∨ y, ¬x ∨ ¬y, ~~z~~} =

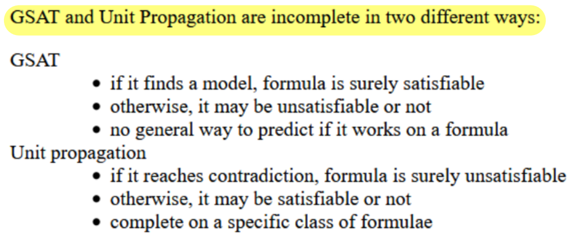
{x ∨ y, x ∨ ¬y, ¬x ∨ y, ¬x ∨ ¬y}

no unit clause

algorithm stops

$\{z = true\}$ and replace it in clauses and we end with formula that has not unit clause. We cannot do anything else, algorithm stops because it is based on the principle of unit clause. The algorithm didn't have definite answer because the resulting formula is not contraddictory. And since the clauses are not all satisfies the algorithm didn't even finish by saying that that the formula is satisfiable because the formula is actually not satisfiable. It is just that Unit Propagation couldn't establish is this formula is satisfiable or not.

If we "try" by our hand to verify if the formula is satisfiable or not we can see that it is not satisfiable but the Unit Propagation cannot say anything because it is based on unit clause and we do not have any unit clause available. Unit Propagation didn't reach a contraddiction. This is one of the reason why Unit Propagation is incomplete.



GSAT and Unit Propagation are incomplete in two different ways:

GSAT
- if it finds a model, formula is surely satisfiable
- otherwise, it may be unsatisfiable or not
- no general way to predict if it works on a formula

Unit propagation
- if it reaches contradiction, formula is surely unsatisfiable
- otherwise, it may be satisfiable or not
- complete on a specific class of formulae

If in GSAT we find a model the formula is satisfiable otherwise we do not know. Instead, in Unit Propagation if we find a contraddiction then the formula is unsatisfiable, otherwise we do not know.

There is one particular case in which Unit Propagation is complete and this is the case of *Horn clause.*
Horn clause is a clause in which there is *at most one* **positive literal**. Horn formula is set of Horn clauses.
Example of Horn clauses:

- $x \vee \neg y$
- $\neg y \vee z \vee \neg w$
- y
- $\neg x$
- $\neg x \vee \neg y \vee \neg z$