Week 7

Why is **SAT** important?

- The first example of a problem shown to be **NP-**complete (recall the Cook-Levin Theorem)
  - All NP-complete problems can be **reduced to SAT**
  - Finding a **polynomial time** algorithm for SAT proves that **P = NP**

- Reduction to SAT are **often straightforward**
  - **Logic acts as a specification language to describe real-world problems**

- Even in **P != NP** we still want to be able to answer **real-worlds problems** as quickly as possible.

- **Example:**
  - We can each of the six statement from earlier with a **propositional formula**

	:	If Bob is telling the truth then so is Connie	$B \rightarrow C$
	:	Either David or Francis is telling the truth	$D \vee F$
	:	Alice and Ellen are both lying	$\neg A \wedge \neg E$
	:	Francis and Alice and both telling the truth	$F \wedge A$
	:	If David is telling the truth then Bob must be lying	$D \rightarrow \neg B$
	:	If Ellen is lying then so is Connie	$\neg E \rightarrow \neg C$

(ther variable A denotes the proposition *"Alice is telling the truth"*, etc.)

  - However, each statement is true **if and only if** the individual uttering it is telling the truth. Therefore

$$\begin{aligned} A &\leftrightarrow (B \rightarrow C) \\ B &\leftrightarrow (D \vee F) \\ C &\leftrightarrow (\neg A \wedge \neg E) \end{aligned} \qquad \begin{aligned} D &\leftrightarrow (F \wedge A) \\ E &\leftrightarrow (D \rightarrow \neg B) \\ F &\leftrightarrow (\neg E \rightarrow \neg C) \end{aligned}$$

  - We are seeking a **satisfying assignment** that makes all of the formulas true at the same time

						Above formulas
???	**???**	**???**	**???**	**???**	**???**	**true**

- However, it is not straightforward to find a **satisfying assignment** for large problems!

n variables $\Longrightarrow$ 2^n rows in truth table!

(270 variables requires more rows than there are atoms in the observable universe!)

- Indeed, we can see that SAT is an **NP-COMPLETE** problem!

- Given the importance of SAT in solving many **real-world problems,** a great deal of research has been invested in finding **efficient algorithms** for solving the satisfiability problem.

Greedy SAT Solving

- **Naïve Greedy Algorithm**

  Step 1) Guess a variable assignment!

  $P :=$ **TRUE**, $Q :=$ **FALSE**, and $R :=$ **FALSE**

  (for example)

  Step 2) Count the number of *satisfied clauses*

  Step 3) Flip the assignment of a variable that leads the the *biggest increase* in the number of satisfied clauses,

  Step 4) Repeat until no further improvements to the 'score' are possible.

- Pick a random variable assignment and aim for the score which satisfies the number of the formulas. In the picture below we stop because all formulas are satisfied which takes only 3 changes.

P	Q	R	$(\neg P \vee Q \vee R)$	$(P \vee Q)$	$(Q \vee R)$	$(\neg P \vee Q)$	$(\neg P \vee \neg Q \vee R)$	$(\neg P \vee \neg Q)$	$(P \vee \neg Q \vee R)$	Score
T	F	F	F	T	F	F	T	T	T	4
T	T	F	T	T	T	T	F	F	T	5
F	T	F	T	T	T	T	T	T	F	6
F	T	T	T	T	T	T	T	T	T	7

(we found a solution with only 3 changes!)

Greedy SAT Solving

- What is the **running time** for the greedy algorithm ?
  - Choosing an initial assignment takes **constant time O(1)**
  - Evaluating the set of clauses takes **linear time O(n)**
  - For each of the ***n*** possible flips, we need to evaluate the set of clauses, this requires **quadratic time O(n^2)**
  - In the worst case we many need to flip all ***n*** assignments. Hence, we must repeat the above step **at most *n*** times!

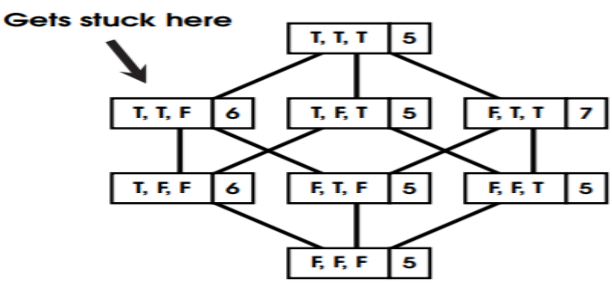
$$T(n) = O(1) + O(n) + n \cdot O(n^2) = O(n^3)$$

$\rightsquigarrow$ **P = NP !!!**

- Unfortunately the algorithm is **incomplete**

  (it does not always find a solution if it exists!)

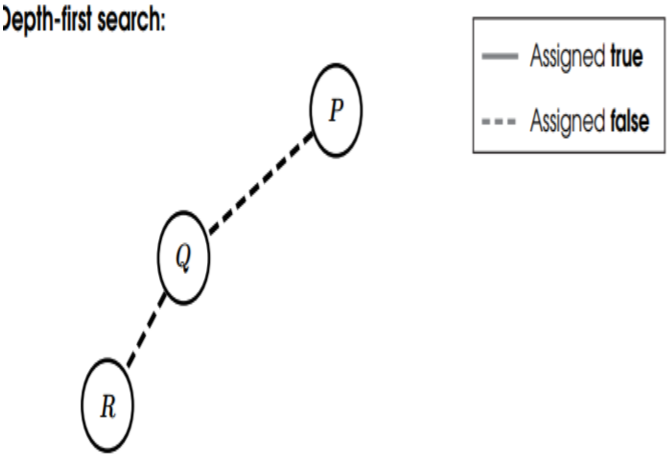
  - **Example:**

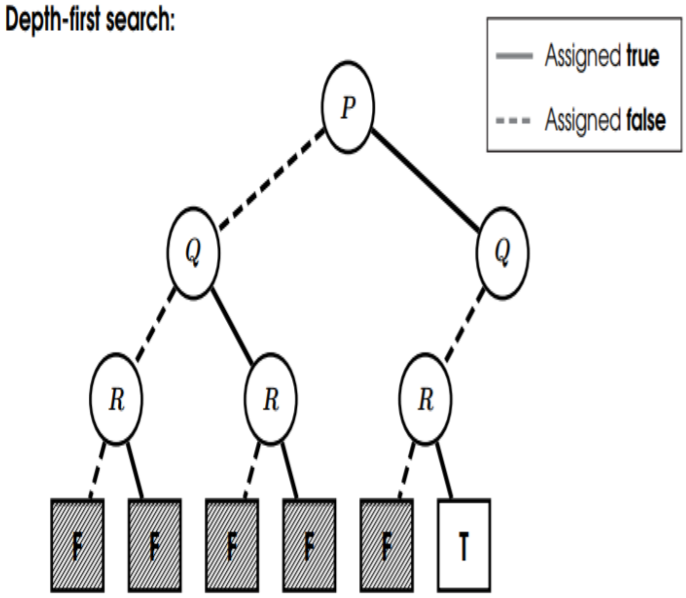


$(P \vee Q)$ ✓
$(P \vee R)$ ✓
$(P \vee \neg Q \vee R)$ ✓
$(\neg P \vee \neg Q)$ ✗
$(\neg P \vee Q)$ ✓
$(\neg P \vee \neg R)$ ✓
$(P \vee Q \vee \neg R)$ ✓

(each edge corresponds to a single assignment flip)

Efficient SAT Solving





$(P \vee Q \vee R)$ $(\neg P \vee Q \vee R)$
$(P \vee \neg R)$ $(P \vee \neg Q)$ $(\neg Q \vee R)$

$(P \vee Q \vee R)$ ✓ $(\neg P \vee Q \vee R)$ ✓
$(P \vee \neg R)$ ✓ $(P \vee \neg Q)$ ✓ $(\neg Q \vee R)$ ✓

Efficient SAT Solving Issues

- **PROBLEMS WITH THIS APPROACH:**
  - May still need to search the **entire tree** for example when there is not correct solution to decide that there is no solution algorithm must search **entire tree.**
  - The tree **grows exponentially** in the number of variables.
  - May end up **duplicating** a lot of the same calculations.

- **HOW TO ADDRESS THESE PROBLEMS:**
  - Unfortunately, we **can't' avoid** having to occasionally search the entire tree, (again unsatisfiable formulas must be fully explored)
  - However, we can attempt to **prune the tree**, so there is less to search!
  -

The DPLL Algorithm

- **The Davis-Putnam-Logemann-Loveland (DPLL) algorithm**
  - A **backtrack search** algorithm, **(similar to DFS)**
  - Employs **pure literal elimination** and **unit propogation**
  - Proposed in 1962 by M.Davis, G.Logemann, and D.Loveland
  - Previous work in 1960 by M.Davis and H.Putnam.
    (the DP algorithm is *resolution-based* – not covered in this course)

Week 7

The DPPL Algorithm

PURE LITERAL ELIMINATION

- A **pure literal** is any literal that occurs only positively or negatively in all clauses
  - (P v Q v -S), (P v -Q v R), (Q v -R v -S) – (**both P and -S are PURE LITERALS)**
  - Therefore, we can always **assign** pure literals without risk of a conflict

$$\boxed{P := \textbf{TRUE} \quad \text{and} \quad S := \textbf{FALSE}}$$

  - We can then **ELIMINATE** any clause containing a ***PURE LITERAL (pure literal elimination assigns literals that SHOULD be assigned)***
  - MUST MAKE SURE A LITERAL REPRESENT ONLY ONE STATE EITHER TRUE OR FALSE IN ALL CLAUSES NOT BOTH. This is because in all clauses the value is the same, so it won't bring conflicts and it will automatically reduce the search space by half.

UNIT PROPAGATION

- A **unit clause** is any clause that contains only a ***SINGLE LITERAL***

$$\neg Q, \quad (P \vee Q), \quad (\neg P \vee \neg R \vee S), \quad (\neg P \vee Q \vee \neg S)$$
$$(\neg P \vee \neg Q), \quad R, \quad (Q \vee R \vee S)$$

(both $\neg Q$ and R is a *unit clauses*)

- We have **no choice** in the assignment of unit clauses - Q := FALSE and R := TRUE
(**unit propagation assigns literals that MUST be assigned)**
- After this step we can **eliminate** any clause containing a **UNIT CLAUSE**

$$\cancel{\neg Q}, \quad (P \vee Q), \quad (\neg P \vee \neg R \vee S), \quad (\neg P \vee Q \vee \neg S)$$
$$\cancel{(\neg P \vee \neg Q)}, \quad \cancel{R}, \quad \cancel{(Q \vee R \vee S)}$$

- We can **SIMPLIFY** any clause containing the negation of the unit clause.

$$\begin{array}{ccccc} (P \vee Q) & , & (\neg P \vee \neg R \vee S) & , & (\neg P \vee Q \vee \neg S) \\ \Downarrow & & \Downarrow & & \Downarrow \\ P & , & (\neg P \vee S) & , & (\neg P \vee \neg S) \end{array}$$

- These assignments **propagate** leading to further elimination (This means after simplifying the expressions we could derive new **UNIT CLAUSES** which should also be removed.

$$\boxed{P := \textbf{TRUE}}$$

(new unit clauses can be assigned)

$$\begin{array}{ccccc} P & , & (\neg P \vee S) & , & (\neg P \vee \neg S) \\ & & \Downarrow & & \Downarrow \\ & & S & , & \neg S \end{array}$$

conflict

Therefore the 7 formulas are unsatisfiable, if you reach conflict u made mistake and you need to backtrack

The DPLL Algorithm

- **The DPPL Algorithm consist of:** DPLL(set of clauses C, partial assignment U)

- **Set of clauses = *C***
  - The set of clauses that we seek to satisfy
- **Partial Assignment = *U***
  - A set of literals that have we already been assigned **TRUE**
  - Initially **empty** to indicate the start of the search.
  - We add new literals to ***U*** as the search progresses. MUST ASSIGNED TO BE TRUE AND SHOULD ASSIGNED TO BE TRUE

Inputs: set of clauses C, partial assignment $U = \emptyset$

Step 1) If $C = \emptyset$ then return **TRUE**

Step 2) If C contains a *conflict* then return **FALSE**,

Step 3) Apply *unit propogation*

- Add any unit clauses to U, and simplify all clauses in C

Step 4) Apply *pure literal elimination*

- Add any pure literals to U, and simplify all clauses in C

Step 5) Choose any unassigned variable P to branch with

Step 5a) If DPLL$(C,\ U \cup \{\neg P\})$ = **TRUE**, then return **TRUE**

Step 5b) Else if DPLL$(C,\ U \cup \{P\})$ = **TRUE**, then return **TRUE**

Step 5c) Else return **FALSE**

- Step one takes care of trivial case when there are no clauses
- If C contains a conflict in clauses, then return **FALSE**
- Apply unit propagation, add a unit clause to **U** (commitment store) keep doing this until you run out of things to do
- Apply pure literal elimination add it to **U** (commitment store) and eliminate from your clause set that contain pure literals.
- This bit is a divide and conquer
  - Choose a variable -**P** add it to ***U*, if I can find evaluation that makes this true, it can make this formula is satisfiable. IF THIS RETURNS TRUE THAT DOES NOT MEAN THE FORMULA IS NOT SATISFIABLE.**
  - Choose a variable **P** add it to ***U***, **if I can find evaluation that makes this true.**
  - IF BOTH BRANCHES RETURN FALSE THEN THE ORIGINAL SET OF FORMULAS ARE FALASE BECAUSE ESSENTIALY THERE ARE ONLY TWO CHOICES FOR P. AND THIS IS BECAUSE THERE IS INHERIT CONTRADICTION IN OUR CLAUSES. THERE IS NO WAY TO PROCEED.
    - *This algorithm produces TRUE or FALSE answer*
- To get the variable assignments values, LOOK INTO ***U*** which keep record of assignments as we go along.

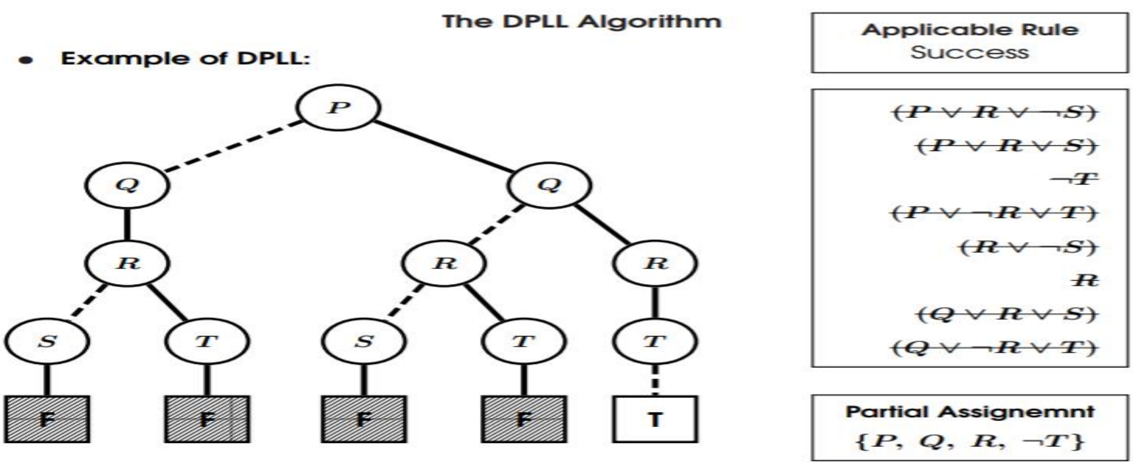
Week 7

<u>The DPPL Algorithm</u>

EXAMPLE

We start from P

1. There are no Unit clauses
2. There is no pure literal
3. Make a branch and chose to make -P to be true
4. We add **-P** into ***U*** and we can remove anything that we made true, so at this point everything containing **-P** is eliminated.
5. **Q** now is a pure literal
6. Add **Q** to ***U*** and eliminate all formulas including **Q**
7. There is no pure literal or unit clause, so we pick random Literal, lets pick **R**
8. Make a branch and chose to make **-R** to be true
9. Eliminate everything that contains **-R**
10. add **-R** to ***U*** partial assignment set
11. This gives us problems because there is contradiction between **S** and **-S**
12. So, we backtrack to **R** and make opposite choice
13. If we make **R** to be true , we bring another conflict between **T** and **-T**
14. **So, whatever is assigned to R will give false**
15. We backtrack to P and not to Q because Q was a pure literal.
16. Etc…
17. Etc..



Alternative Presentation:

Rule	Partial Assignment	$(P \vee R \vee \neg S)$	$(P \vee R \vee S)$	$(\neg R \vee \neg T)$	$(P \vee \neg R \vee T)$	$(\neg P \vee R \vee \neg S)$	$(\neg P \vee \neg Q \vee R)$	$(Q \vee R \vee S)$	$(Q \vee \neg R \vee T)$
Initial	$\emptyset$								
Branch	$\neg P$					✓	✓		
Pure Literal	$\neg P, Q$					✓	✓	✓	✓
Branch	$\neg P, Q, \neg R$	✗	✗	✓	✓	✓	✓	✓	✓
Backtrack	$\neg P, Q, R$	✓	✓	✗	✗	✓	✓	✓	✓
Backtrack	P	✓	✓		✓				
Branch	$P, \neg Q$	✓	✓		✓		✓		

Alternative Presentation (cont.):

Rule	Partial Assignment	$(P \vee R \vee \neg S)$	$(P \vee R \vee S)$	$(\neg R \vee \neg T)$	$(P \vee \neg R \vee T)$	$(\neg P \vee R \vee \neg S)$	$(\neg P \vee \neg Q \vee R)$	$(Q \vee R \vee S)$	$(Q \vee \neg R \vee T)$
Cont.	$P, \neg Q$	✓	✓		✓		✓		
Branch	$P, \neg Q, \neg R$	✓	✓	✓	✓	✗	✓	✗	✓
Backtrack	$P, \neg Q, R$	✓	✓	✗	✓	✓	✓	✓	✗
Backtrack	P, Q	✓	✓		✓			✓	✓
Unit Prop.	P, Q, R	✓	✓		✓		✓	✓	✓
Unit Prop.	$P, Q, R, \neg T$	✓	✓	✓	✓	✓	✓	✓	✓

Easy instances of SAT

NSAT WHERE N IS NUMBER OF LITERALS IN A CLAUSE

SAT IS THE HARDEST INSTANCE

- *All the NSAT problems >= 3SAT are NP-COMPLETE – they difficulty level is the same. If we solve SAT, we can solve all.*

- **The NSAT problem:**
  - **Input)** A propositional formula ***F*** such that:
    - ***F*** is in **Conjunctive Normal Form (CNF)**
    - Each clause of ***F*** contains **at most *N*** literals.
  - **Output)**
    - Decide if the formula ***Fi*** is satisfiable.

- *It is easy to **reduce** one version of **SAT** to the next* $\boxed{\text{2SAT} \leq_p \text{3SAT} \leq_p \text{4SAT} \leq_p \text{5SAT} \leq_p \dots \leq_p \text{SAT}}$

- Also, we can find a reduction that goes the **other way** $\boxed{\text{SAT} \leq_p \dots \leq_p \text{5SAT} \leq_p \text{4SAT} \leq_p \text{3SAT}}$

- It is enough to show that $\boxed{\text{SAT} \leq_p \text{3SAT}}$
  (since then we have (N+1)SAT $\leq_p$ SAT $\leq_p$ 3SAT $\leq_p$ NSAT)

- We need a **reduction** that converts any *CNF* formula into a *CNF* formula in which each clause contains at most 3 literals!

SAT POLYNOMIAL REDUCTION TO 3SAT

Theorem SAT is polynomially reducible to 3SAT.

Proof:

Step 1) Find a clause which contains **more than 3 literals**

$$(P \vee \neg Q \vee R \vee S) \wedge (Q \vee \neg R \vee \neg T)$$
✗ ✓

Step 2) Break up the clause into two pieces

$$(P \vee \neg Q \quad R \vee S) \wedge (Q \vee \neg R \vee \neg T)$$

Step 3) Introduce fresh propositional variable which is used to **BRIDGE THE GAP** (new variable does not bring any new info into formula)

$$(P \vee \neg Q \vee X) \wedge (\neg X \vee R \vee S) \wedge (Q \vee \neg R \vee \neg T)$$

(it is important that X does not occur in the original formula!)

This is equivalent to the original formula because of the following identify

$$(A \vee B) \equiv (A \vee X) \wedge (\neg X \vee B)$$

Step 4) Repeat until all clauses contain **at most 3 literals**

The resultant formula is *satisfiable* if and only if the with the original formula is satisfiable!

Easy instances of SAT

- However, we **CANNOT** use this trick to reduce $\boxed{\text{SAT} \leq_p \text{2SAT}}$

- Consider a single clause with 3 literals $\boxed{(P \lor \neg Q \lor R)}$

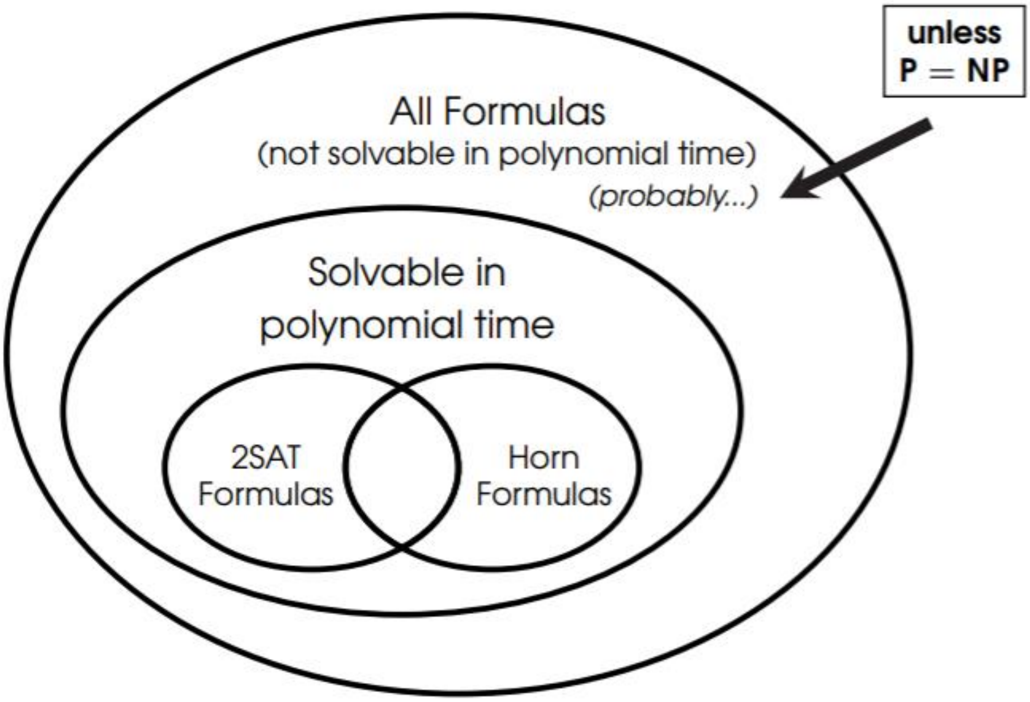
- We can break the clause into two pieces however we end up like $(P \lor \neg Q$ 💥 $R)$

- However, by introducing a fresh variable, we run into problems, because then if we want to introduce another bridge variable and we already have one we would end up with having to bridge variables together and not including the important ones. **INFINITE CYCLE**

$$\boxed{(P \lor \neg Q \lor X) \ \land \ (\neg X \lor R)}$$

(this formula is equivalent but we are still stuck with 3 literals!)

Breakdown of All Propositional formulas



Solving the 2SAT Problem

Theorem	2SAT is solvable in polynomial time.

Proof: We can reduce 2SAT to the **strongly connected component** problem for directed graphs.

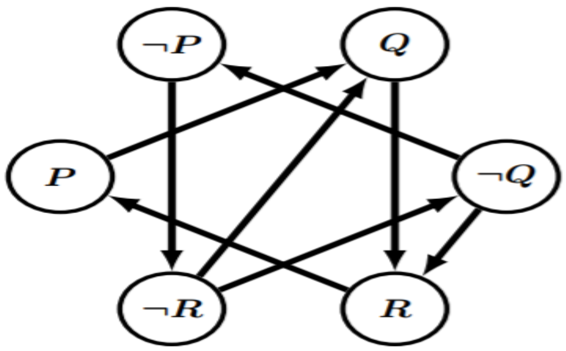
Step 1) Write each caluse as **two implications (using this rule -->)** $\boxed{(A \vee B) \equiv (\neg A \rightarrow B) \wedge (\neg B \rightarrow A)}$

Example:

$$\begin{array}{l} (\neg P \vee Q) \\ (\neg Q \vee R) \\ (P \vee \neg R) \\ (R \vee Q) \end{array} \Longrightarrow \begin{array}{lcl} (P \rightarrow Q) & \wedge & (\neg Q \rightarrow \neg P) \\ (Q \rightarrow R) & \wedge & (\neg R \rightarrow \neg Q) \\ (\neg P \rightarrow \neg R) & \wedge & (R \rightarrow P) \\ (\neg R \rightarrow Q) & \wedge & (\neg Q \rightarrow R) \end{array}$$

REMEMBERING THAT SCC IS A CYCLE A PATH FROM ONE TO ANOTHER

Step 2) Construct the **implication graph**

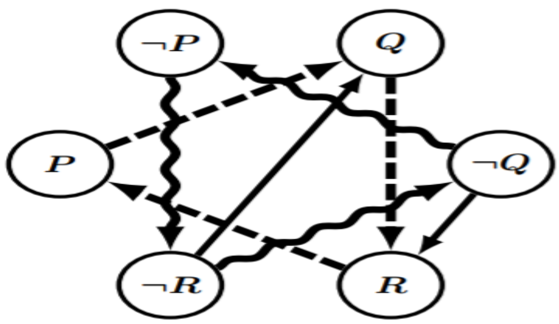


$(P \rightarrow Q)$
$(\neg Q \rightarrow \neg P)$
$(Q \rightarrow R)$
$(\neg R \rightarrow \neg Q)$
$(\neg P \rightarrow \neg R)$
$(R \rightarrow P)$
$(\neg R \rightarrow Q)$
$(\neg Q \rightarrow R)$

Step 3) F is satisfiable iff every **strongly connected component** is consistent.
(consistent = does not contain a literal and its negation)

TWO FIND SCC WE LOOK FOR THE CYCLES SO -P, -R, -Q or P, Q, R

Step 2) Construct the **implication graph**



Strongly Connected Components
$\{ P, Q, R \}$
&
$\{ \neg P, \neg Q, \neg R \}$

Step 3) F is satisfiable iff every **strongly connected component** is consistent.
(consistent = does not contain a literal and its negation)

THEREFORE THIS FORMULA IS SATISFIABLE

Week 7

Solving the 2SAT Problem

Conclusion) The reducton can be performed in **polynomial time,** therefore

$$\boxed{\text{2SAT} \leq_p \text{SCC}}$$

(where SCC denotes the strongly connected component problem)

However, the **SCC** problem is decidable in **polynomial time** (in fact, even in linear time!)

Therefore **2SAT is in CLASS P!** $\boxed{\Rightarrow \quad \textbf{2SAT is in P!}}$

Summary

- **Every formula** with at most 2 literals per clause can be decided in **(deterministic) polynomial time.**

- There are **some formulas** with 3 literals per clause that cannot be deided in POLYNOMIAL TIME! **(unless P = NP … we don't know!)**

- Is Every Formula with 3 literals per caluse is hard to solve ? No!

- **Propositional Horn Clauses:**

$$\boxed{P,\ (\neg P \vee Q),\ (\neg P \vee \neg Q \vee R),\ (\neg Q \vee S),\ (\neg S \vee \neg R \vee T)}$$

(each clause contains at most one positive literal)

Can be solved with just Unit Propogation and Pure Literal Elimination

Why Greedy SAT Solving is incomplete ?

It is incomplete because there could exist a soltuion that Greedy SAT Algorithm would not found. It can happen when flipping variables greedily traps us in a cycle such as l -> l' -> l'' -> l