5CCS2FC2: Foundations of Computing II

Efficient SAT Solving

Week 7

Dr Christopher Hampson

Department of Informatics

King's College London

Warm-up : Who is telling the truth?

- Consider the six statements made by the following six individuals.

Alice	:	If Bob is telling the truth then so is Connie,
Bob	:	Either David or Francis is telling the truth,
Connie	:	Alice and Ellen are both lying,
David	:	Francis and Alice and both telling the truth,
Ellen	:	If David is telling the truth then Bob must be lying,
Francis	:	If Ellen is lying then so is Connie.

Objectives for Today

- To be able to perform the **Greedy SAT Algorithm**
    - and to be able to explain why it is **incompete**.
- To be able to perform the **DPLL Algorithm** to solve (small) instances of SAT
- To solve easy instances of SAT using the **implication graph**

Why is SAT important?

Why is SAT important?

- The first example of a problem shown to be **NP-complete**

  (recall the Cook-Levin Theorem from week 4)

  - All NP-complete problems can be **reduced to SAT**
  - Finding a **polynomial time** algorithm for SAT proves that $\mathbf{P} = \mathbf{NP}$

- Reductions to SAT are **often straightforward**

  (logic acts as a *specification language* to describe real-world problems)

- Even if $\mathbf{P} \neq \mathbf{NP}$ we still want to be able to answer **real-world problems** as quickly as possible

Why is SAT important?

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

	: If Bob is telling the truth then so is Connie	$B \to C$
	: 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	$D \to \neg B$
	: If Ellen is lying then so is Connie	$\neg E \to \neg C$

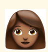
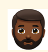
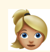
(ther variable A denotes the proposition *"Alice is telling the truth"*, etc.)

Why is SAT important?

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

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

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

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

Why is SAT important?

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

$$\boxed{n \text{ variables} \quad \Longrightarrow \quad 2^n \text{ rows in truth table!}}$$

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

- Indeed, we know that SAT is an **NP-complete** problem!

- Given the importance of SAT is 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

Greedy SAT Solving

- **Naïve Greedy Algorithm**

Step 1) Guess a variable assignment!

$$\boxed{P := \textsf{TRUE}, \quad Q := \textsf{FALSE}, \quad \text{and} \quad R := \textsf{FALSE}}$$

(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.

Greedy SAT Solving

- **Example:**

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** $O(1)$,

    (*e.g.*, we could choose all false, by default)
  - Evaluating the set of clauses takes **linear time** $O(n)$,
  - For each of the n possible flips, we need to evaulate the set of clauses. This requires **quadratic time** $O(n^2)$.
  - In the worst case we may need to flip all n assignements.

    Hence we must repeat the above step **at most** n times!

$$\boxed{T(n) \;=\; O(1) + O(n) + n \cdot O(n^2) \;=\; O(n^3)}$$

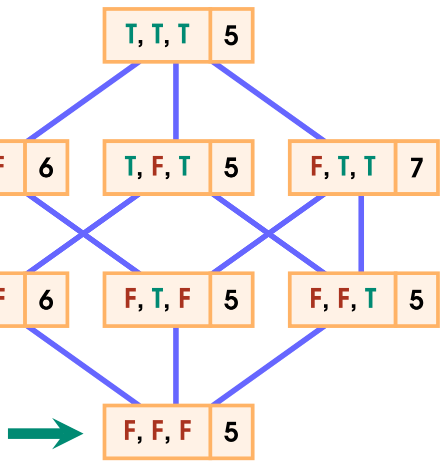
$\rightsquigarrow$ **P $=$ NP !!!**

Greedy SAT Solving

- Unfortunately the algorithm is **incomplete**

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

  - **Example:**



Initial guess

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

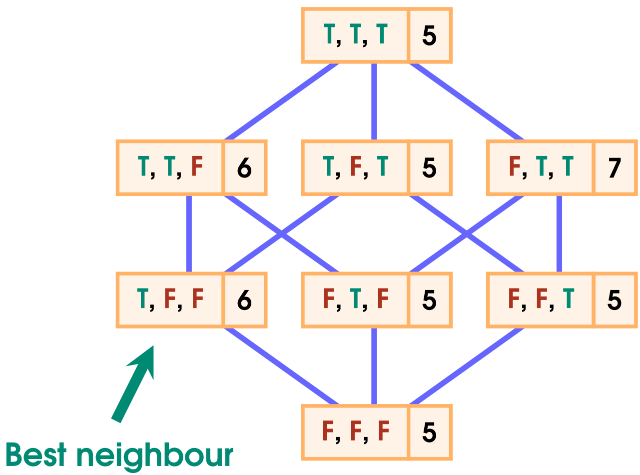
(each edge corresponds to a single assignment flip)

Greedy SAT Solving

- Unfortunately the algorithm is **incomplete**

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

- **Example:**



$(P \vee Q)$ ✓

$(P \vee R)$ ✓

$(P \vee \neg Q \vee R)$ ✓

$(\neg P \vee \neg Q)$ ✓

$(\neg P \vee Q)$ ✗

$(\neg P \vee \neg R)$ ✓

$(P \vee Q \vee \neg R)$ ✓

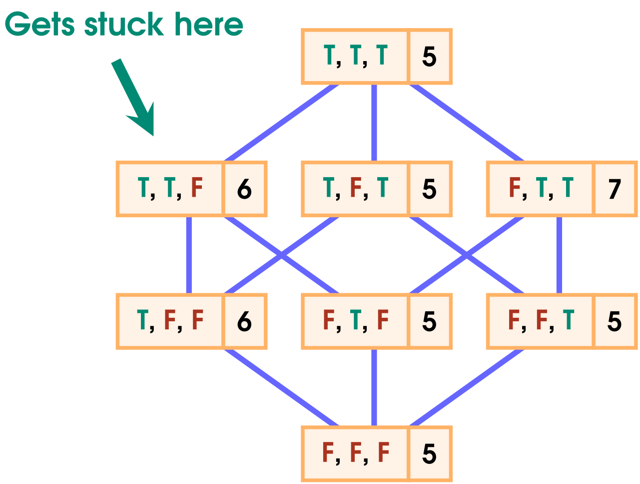
(each edge corresponds to a single assignment flip)

Greedy SAT Solving

- Unfortunately the algorithm is **incomplete**

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

- **Example:**



$(P \lor Q)$ ✓

$(P \lor R)$ ✓

$(P \lor \neg Q \lor R)$ ✓

$(\neg P \lor \neg Q)$ ✗

$(\neg P \lor Q)$ ✓

$(\neg P \lor \neg R)$ ✓

$(P \lor Q \lor \neg R)$ ✓

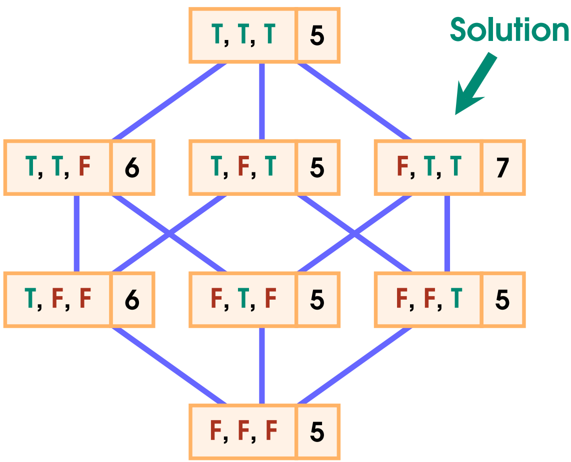
(each edge corresponds to a single assignment flip)

Greedy SAT Solving

- Unfortunately the algorithm is **incomplete**

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

- **Example:**



$(P \lor Q)$ ✓

$(P \lor R)$ ✓

$(P \lor \neg Q \lor R)$ ✓

$(\neg P \lor \neg Q)$ ✓

$(\neg P \lor Q)$ ✓

$(\neg P \lor \neg R)$ ✓

$(P \lor Q \lor \neg R)$ ✓

(each edge corresponds to a single assignment flip)

Efficient SAT Solving

Efficient SAT Solving

- **Depth-first search:**

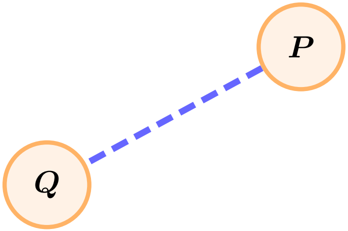
P

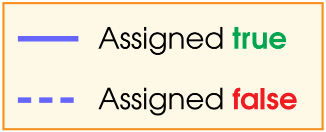
$$(P \vee Q \vee R) \qquad (\neg P \vee Q \vee R)$$

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

Efficient SAT Solving

- **Depth-first search:**



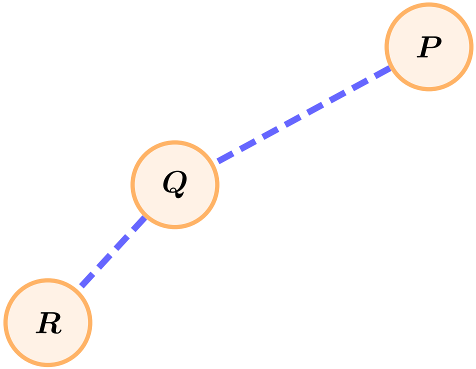


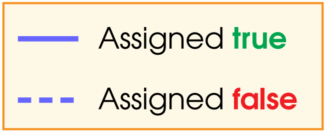
$$(P \vee Q \vee R) \qquad (\neg P \vee Q \vee R)$$

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

Efficient SAT Solving

- **Depth-first search:**



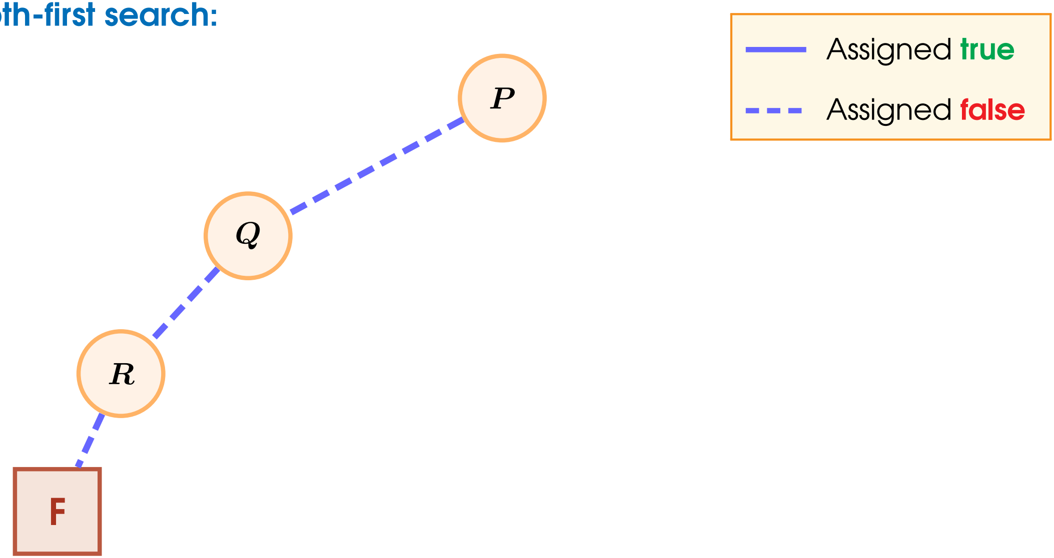


$(P \lor Q \lor R)$ $\quad$ $(\neg P \lor Q \lor R)$

$(P \lor \neg R)$ $\quad$ $(P \lor \neg Q)$ $\quad$ $(\neg Q \lor R)$

Efficient SAT Solving

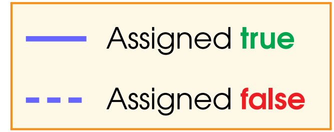
- **Depth-first search:**

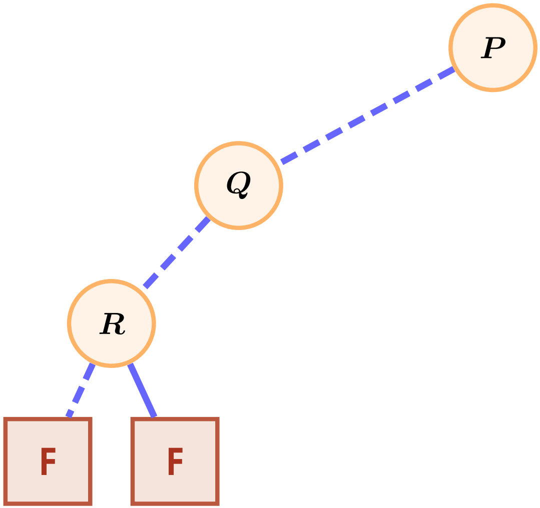


$(P \lor Q \lor R)$ ✗ $(\neg P \lor Q \lor R)$ ✓

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

- **Depth-first search:**



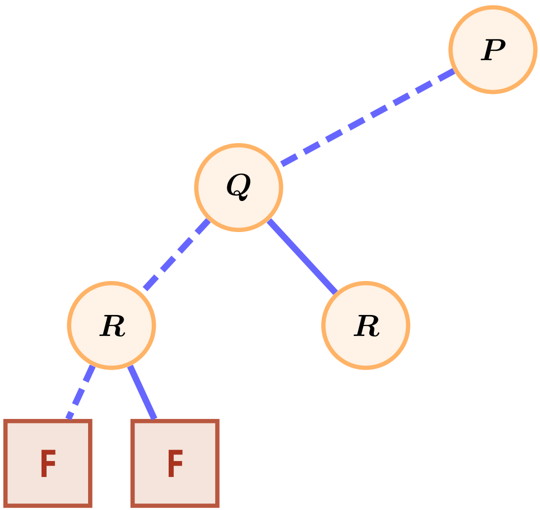


$(P \vee Q \vee R)$ ✓ $\quad (\neg P \vee Q \vee R)$ ✓

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

Efficient SAT Solving

- **Depth-first search:**

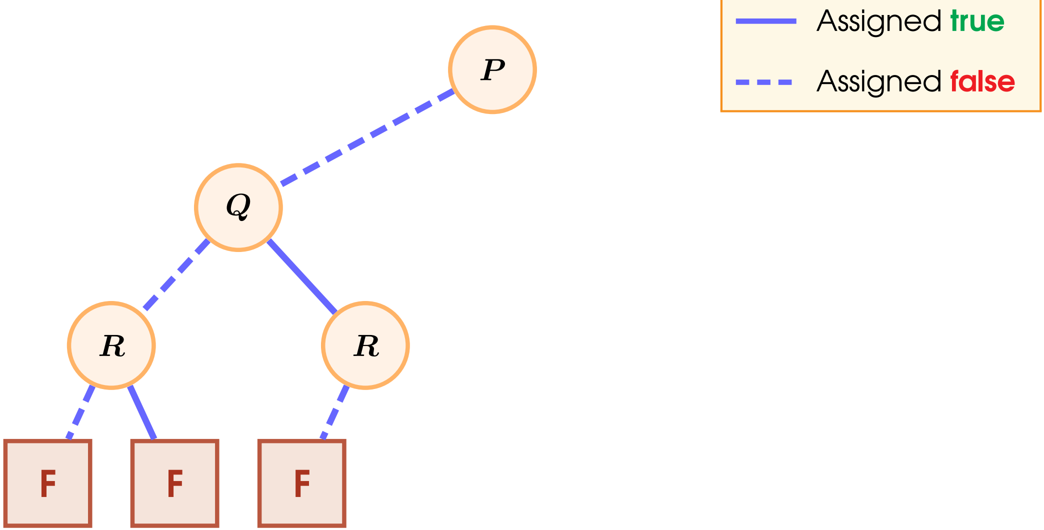


Assigned **true** (solid line)
Assigned **false** (dashed line)

$(P \lor Q \lor R)$ ✓ $\quad (\neg P \lor Q \lor R)$ ✓

$(P \lor \neg R)$ ✗ $\quad (P \lor \neg Q)$ ✓ $\quad (\neg Q \lor R)$ ✓

- **Depth-first search:**

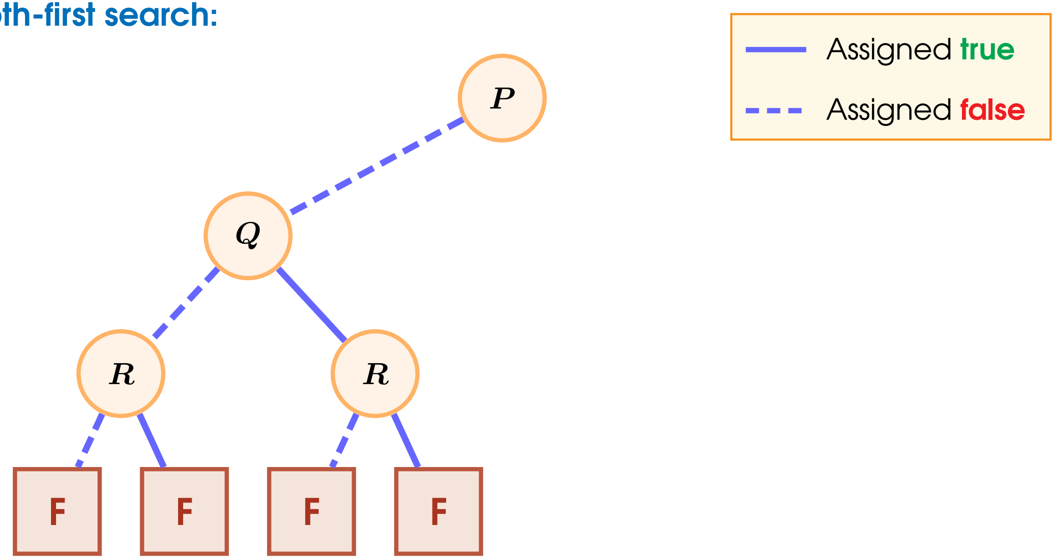


$$(P \vee Q \vee R) \checkmark \qquad (\neg P \vee Q \vee R) \checkmark$$

$$(P \vee \neg R) \checkmark \qquad (P \vee \neg Q) \text{ ✗} \qquad (\neg Q \vee R) \text{ ✗}$$

Efficient SAT Solving

- **Depth-first search:**

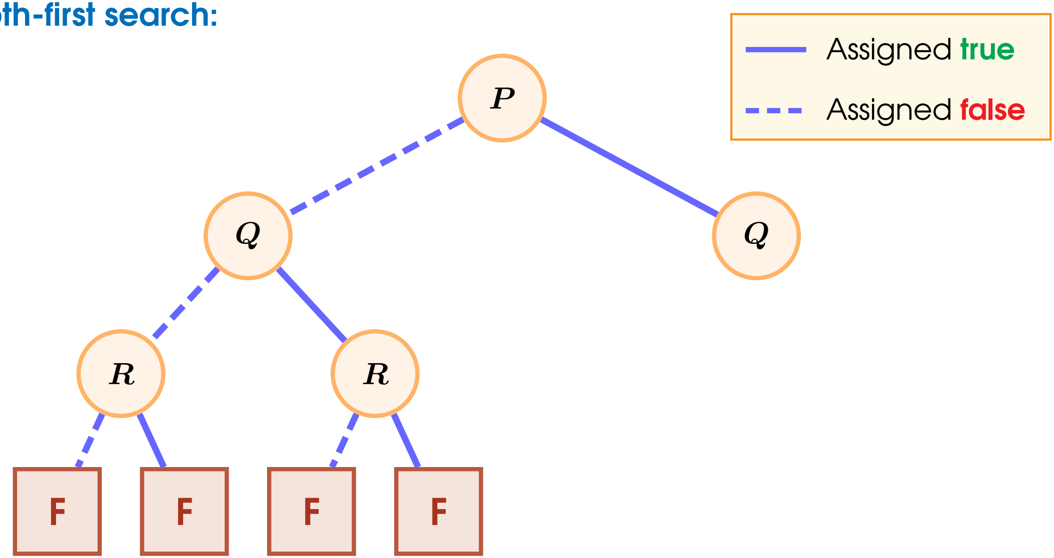


$(P \vee Q \vee R)$ ✓ $(\neg P \vee Q \vee R)$ ✓

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

Efficient SAT Solving

- **Depth-first search:**

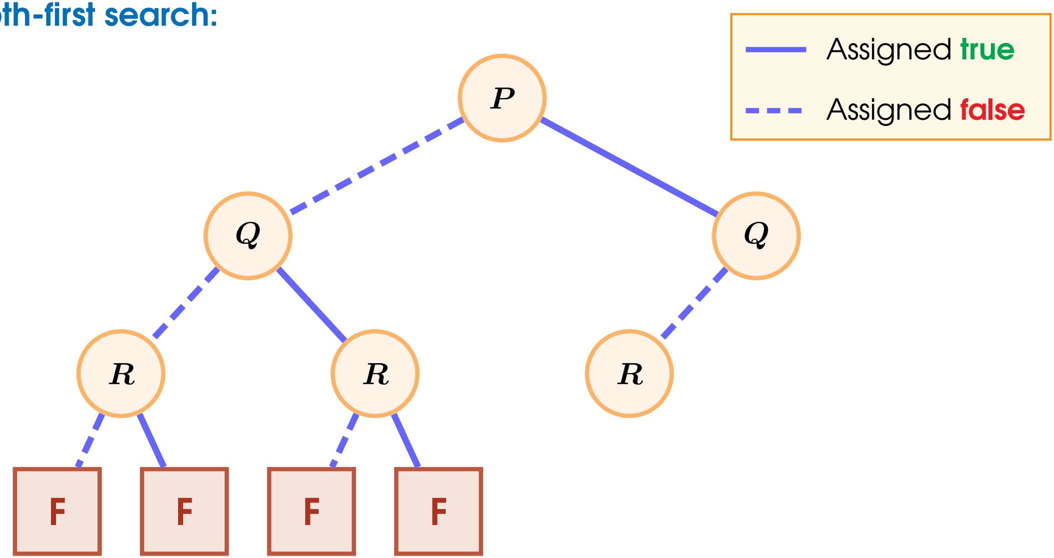


$(P \vee Q \vee R)$ ✓ $(\neg P \vee Q \vee R)$ ✓

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

Efficient SAT Solving

- **Depth-first search:**

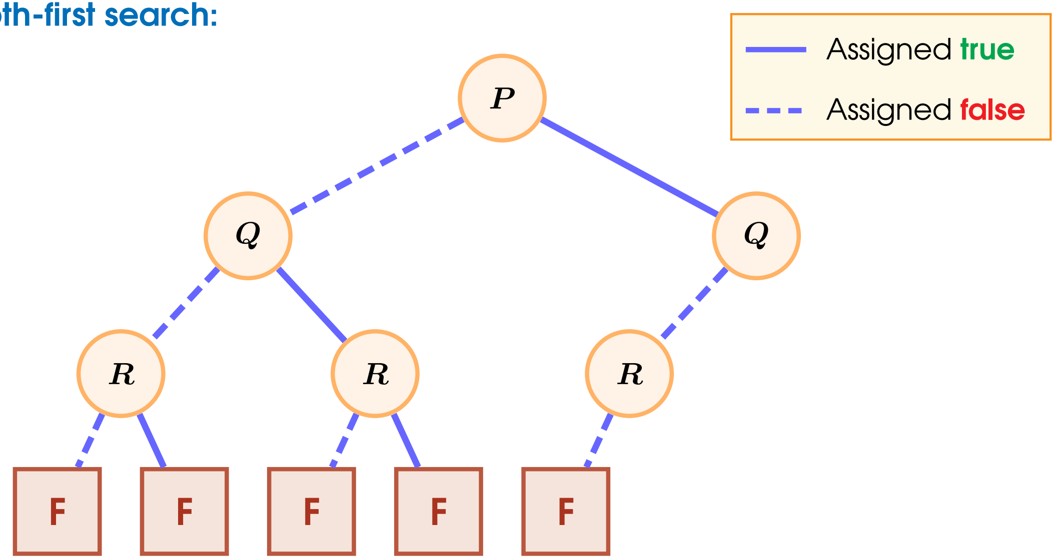


$(P \vee Q \vee R)$ ✓ $(\neg P \vee Q \vee R)$ ✓

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

Efficient SAT Solving

- **Depth-first search:**

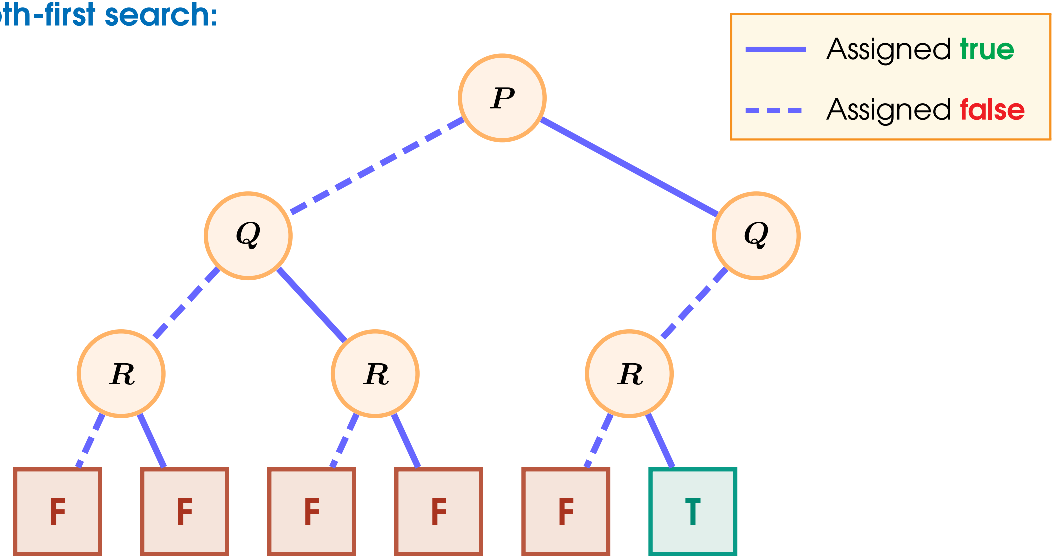


$(P \lor Q \lor R)$ ✓ $(\neg P \lor Q \lor R)$ ✗

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

Efficient SAT Solving

- **Depth-first search:**



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

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

Efficient SAT Solving

- **Problems with this approach**
  - May still need to search the **entire tree**,
  - The tree **grows exponentialy** 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,

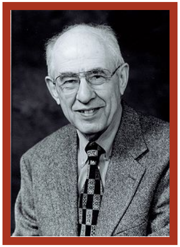
    (unsatisfiable formulas must be *fully* explored)
  - However, we can attempt to **prune the tree**, so there is less to search!

The DPLL Algorithm

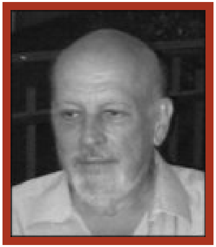
- **The Davis-Putnam-Logemann-Loveland (DPLL) algorithm**
  - A **backtrack search** algorithm, **(similar to DFS)**
  - Employs **pure literal elimination** and **unit propogation**
  - Proposed in 1962 by M.Davis, G.Logemann, and D.Loveland
  - Previous work in 1960 by M.Davis and H.Putnam.

    (the DP algorithm is *resolution-based* – not covered in this course)

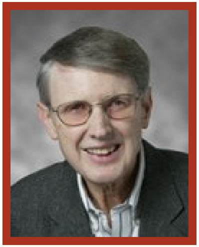
M.Davis



H.Putnam



G.Logemann



D.Loveland

The DPLL Algorithm

- **Pure Literal Elimination:**
  - A **pure literal** is any literal that occurs only positively or negatively in all cluases .

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

(both P and $\neg S$ are *pure literals*)

  - We can always **assign** pure literals without risk of a conflict

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

  - We can then **eliminate** any clause containing a pure literal.

(pure literal elimination assigns literals that SHOULD be assigned)

The DPLL Algorithm

- **Unit Propogation:**
  - A **unit clause** is any 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 propogation assigns literals that MUST be assigned)

- **Unit Propogation (cont.):**
  - 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 neation of the unit clause

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

The DPLL Algorithm

- **Unit Propogation (cont.):**
  - These assignments **propogate** leading to further elimination,

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

(new unit clauses can be assigned)

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

$\rightsquigarrow$ **Conflict!**

- **The DPLL Algorithm**

$$\boxed{\text{DPLL(set of clauses } C, \text{ partial assignment } U \text{)}}$$

- **Set of clauses** C
  - The set we 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,

The DPLL Algorithm

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 $\text{DPLL}(C, U \cup \{\neg P\}) =$ **TRUE**, then return **TRUE**

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

Step 5c) Else return **FALSE**

The DPLL Algorithm

- **Example of DPLL:**

P

Applicable Rule

Branch on P

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

$(P \vee R \vee S)$

$(\neg R \vee \neg T)$

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

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

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

$(Q \vee R \vee S)$

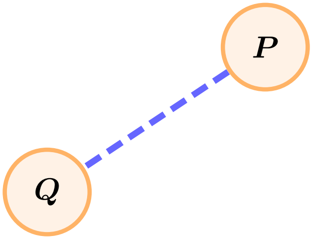
$(Q \vee \neg R \vee T)$

Partial Assignemnt

$\emptyset$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Pure Literal Q

$(R \lor \neg S)$

$(R \lor S)$

$(\neg R \lor \neg T)$

$(\neg R \lor T)$

$\cancel{(\neg P \lor R \lor \neg S)}$

$\cancel{(\neg P \lor \neg Q \lor R)}$

$(Q \lor R \lor S)$

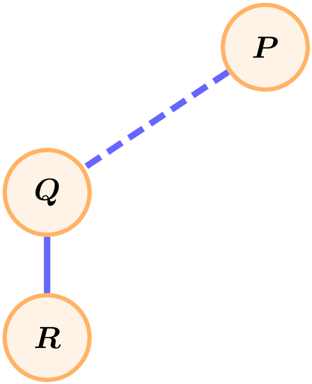
$(Q \lor \neg R \lor T)$

Partial Assignemnt

$\{\neg P\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Branch on R

$(R \vee \neg S)$

$(R \vee S)$

$(\neg R \vee \neg T)$

$(\neg R \vee T)$

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

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

$\cancel{(Q \vee R \vee S)}$

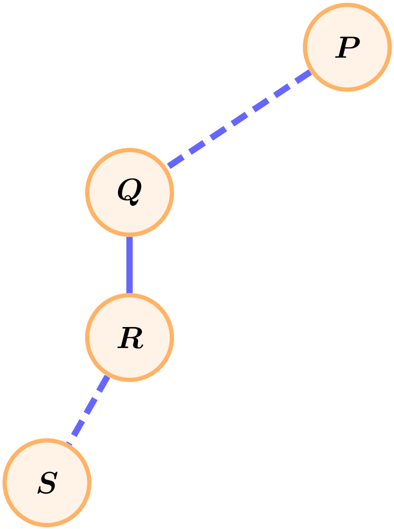
$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt

$\{\neg P,\ Q\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Conflict

$\neg S$

S

$\cancel{(\neg R \vee \neg T)}$

$\cancel{(\neg R \vee T)}$

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

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

$\cancel{(Q \vee R \vee S)}$

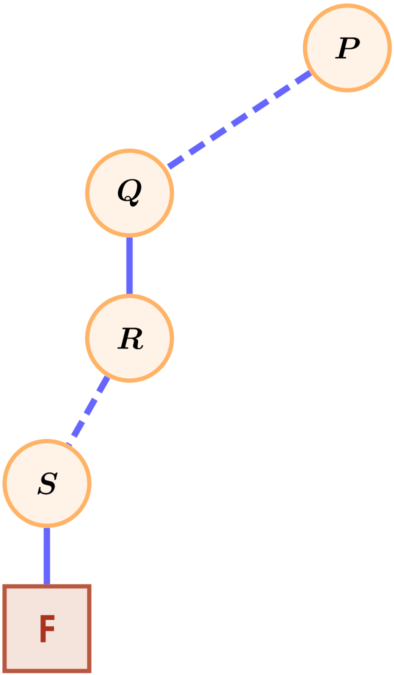
$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt

$\{\neg P,\ Q,\ \neg R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Backtrack to R

$\neg S$

S

$\cancel{(\neg R \vee \neg T)}$

$\cancel{(\neg R \vee T)}$

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

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

$\cancel{(Q \vee R \vee S)}$

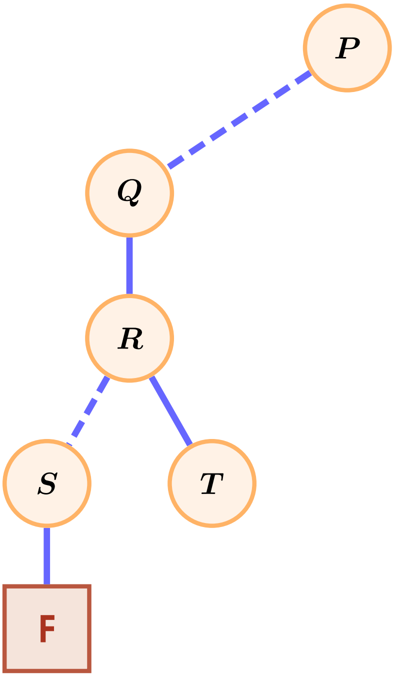
$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt

$\{\neg P,\ Q,\ \neg R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Conflict

$\cancel{(R \vee \neg S)}$

$\cancel{(R \vee S)}$

$\neg T$

T

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

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

$\cancel{(Q \vee R \vee S)}$

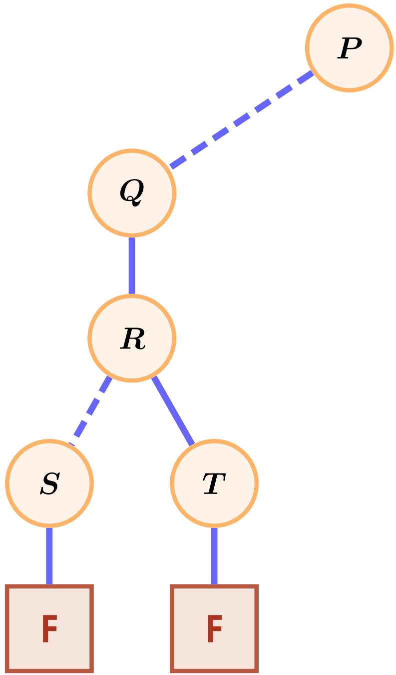
$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt

$\{\neg P,\ Q,\ R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Backtrack to P

$\cancel{(R \vee \neg S)}$

$\cancel{(R \vee S)}$

$\neg T$

T

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

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

$\cancel{(Q \vee R \vee S)}$

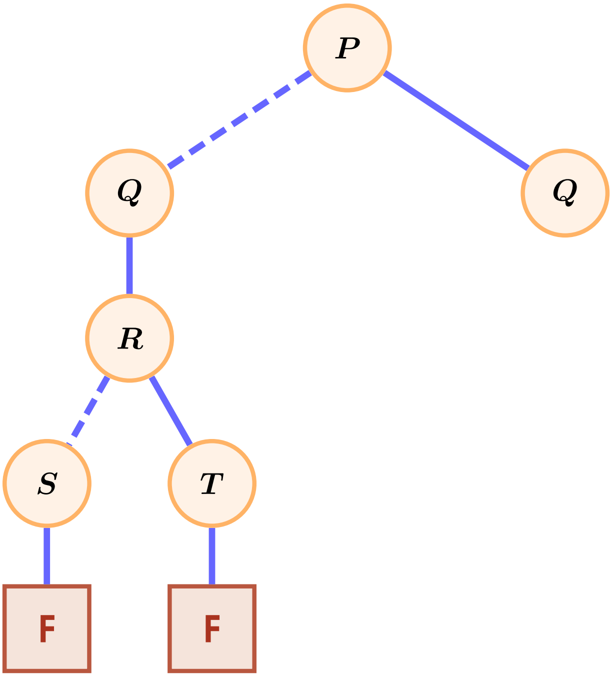
$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt

$\{\neg P,\ Q,\ R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Brach on Q

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

$\cancel{(P \vee R \vee S)}$

$(\neg R \vee \neg T)$

$\cancel{(P \vee \neg R \vee T)}$

$(R \vee \neg S)$

$(\neg Q \vee R)$

$(Q \vee R \vee S)$

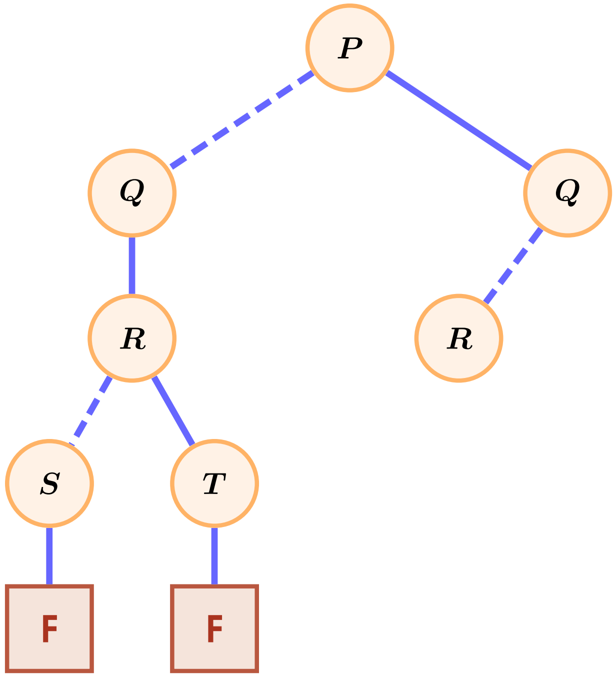
$(Q \vee \neg R \vee T)$

Partial Assignemnt

$\{P\}$

Applicable Rule
Branch on R

- **Example of DPLL:**



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

$\cancel{(P \vee R \vee S)}$

$(\neg R \vee \neg T)$

$\cancel{(P \vee \neg R \vee T)}$

$(R \vee \neg S)$

$\cancel{(\neg Q \vee R)}$

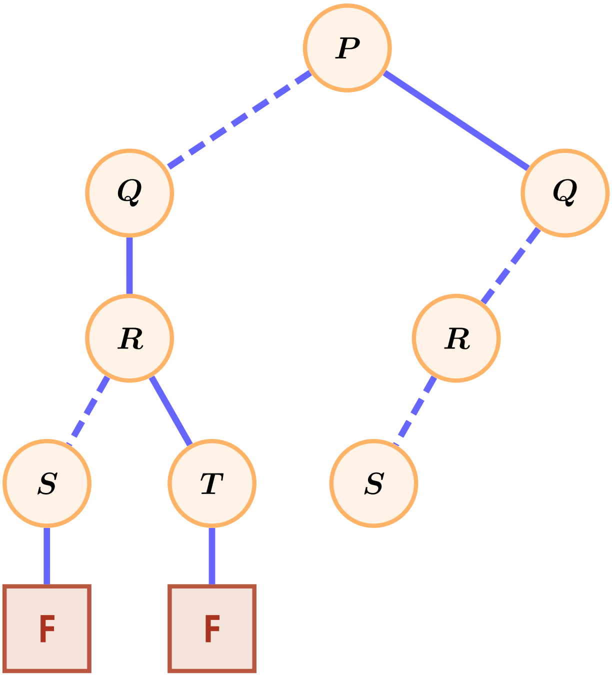
$(R \vee S)$

$(\neg R \vee T)$

Partial Assignemnt
$\{P, \neg Q\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Conflict

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

$\cancel{(P \vee R \vee S)}$

$\cancel{(\neg R \vee \neg T)}$

$\cancel{(P \vee \neg R \vee T)}$

$\neg S$

$\cancel{(\neg Q \vee R)}$

S

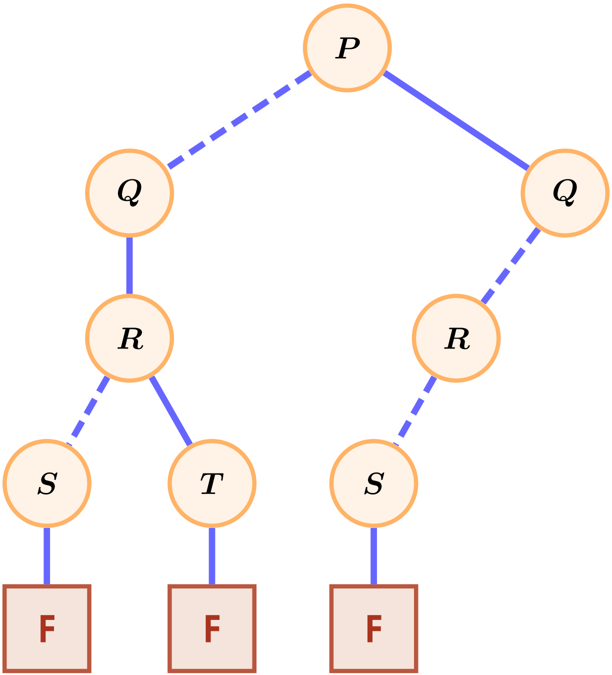
$\cancel{(\neg R \vee T)}$

Partial Assignemnt

$\{P, \neg Q, \neg R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Backtrack

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

$\cancel{(P \vee R \vee S)}$

$\cancel{(\neg R \vee \neg T)}$

$\cancel{(P \vee \neg R \vee T)}$

$\neg S$

$\cancel{(\neg Q \vee R)}$

S

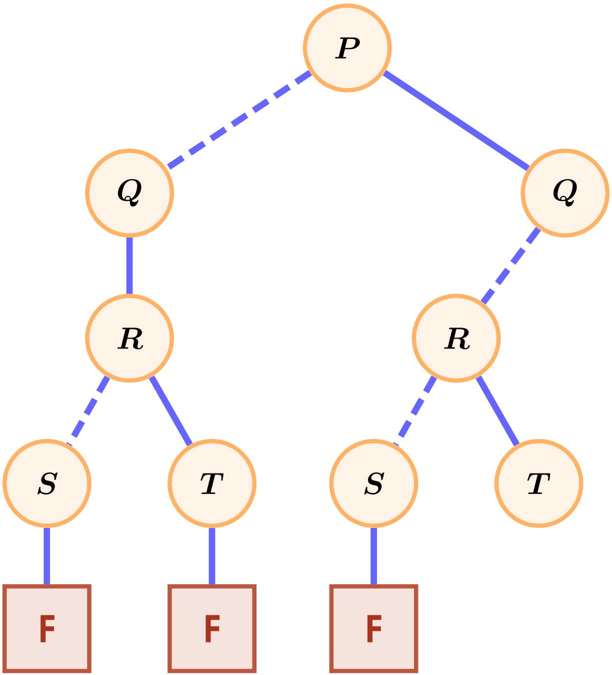
$\cancel{(\neg R \vee T)}$

Partial Assignemnt

$\{P, \neg Q, \neg R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Conflict

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

$\cancel{(P \vee R \vee S)}$

$\neg T$

$\cancel{(P \vee \neg R \vee T)}$

$\cancel{(R \vee \neg S)}$

$\cancel{(\neg Q \vee R)}$

$\cancel{(R \vee S)}$

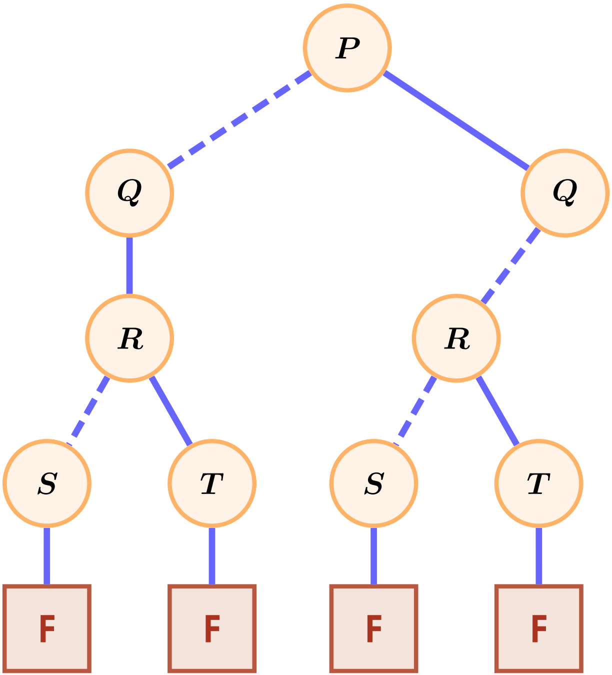
T

Partial Assignemnt

$\{P,\ \neg Q,\ R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Backtrack

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

$\cancel{(P \vee R \vee S)}$

$\neg T$

$\cancel{(P \vee \neg R \vee T)}$

$\cancel{(R \vee \neg S)}$

$\cancel{(\neg Q \vee R)}$

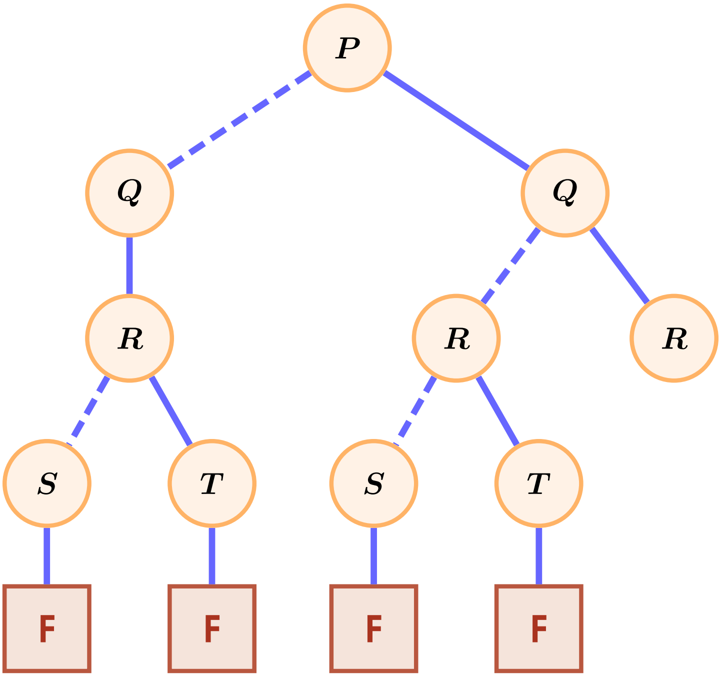
$\cancel{(R \vee S)}$

T

Partial Assignemnt

$\{P,\ \neg Q,\ R\}$

- **Example of DPLL:**



Applicable Rule
Unit Clause

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

$\cancel{(P \vee R \vee S)}$

$(\neg R \vee \neg T)$

$\cancel{(P \vee \neg R \vee T)}$

$(R \vee \neg S)$

R

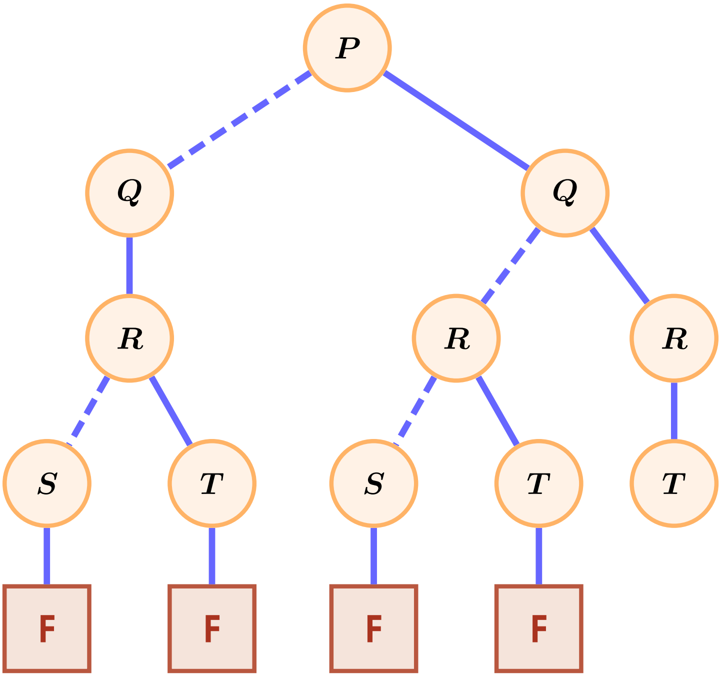
$\cancel{(Q \vee R \vee S)}$

$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt
$\{P,\ Q\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Unit Clause

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

$\cancel{(P \vee R \vee S)}$

$\neg T$

$\cancel{(P \vee \neg R \vee T)}$

$\cancel{(R \vee \neg S)}$

$\cancel{R}$

$\cancel{(Q \vee R \vee S)}$

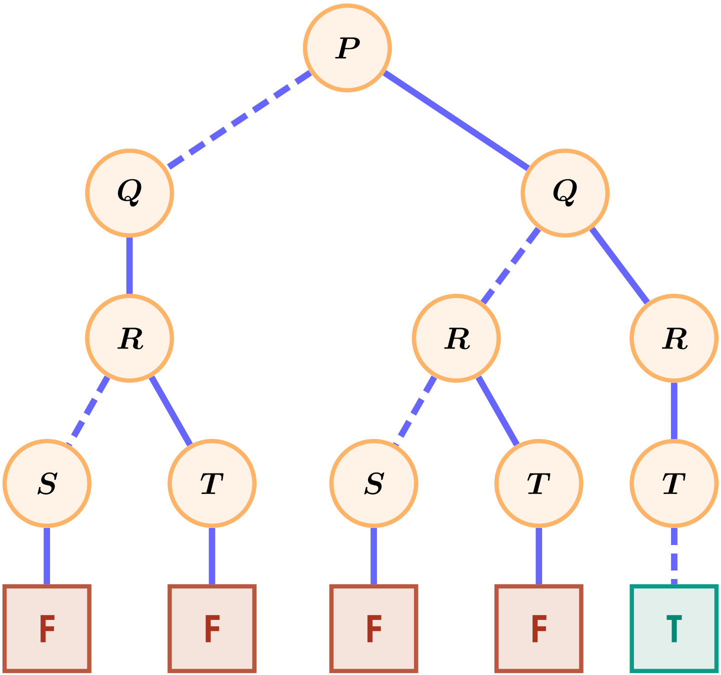
$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt

$\{P,\ Q,\ R\}$

The DPLL Algorithm

- **Example of DPLL:**



Applicable Rule

Success

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

$\cancel{(P \vee R \vee S)}$

$\cancel{\neg T}$

$\cancel{(P \vee \neg R \vee T)}$

$\cancel{(R \vee \neg S)}$

$\cancel{R}$

$\cancel{(Q \vee R \vee S)}$

$\cancel{(Q \vee \neg R \vee T)}$

Partial Assignemnt

$\{P,\ Q,\ R,\ \neg T\}$

The DPLL Algorithm

- **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.	$P, Q, R, \neg T$	✓	✓	✓	✓	✓	✓	✓	✓

Are all SAT problems hard?

'Easy' instances of SAT

- **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 F is satisfiable.

- It is easy to **reduce** one version of SAT to the next

$$\boxed{\text{2SAT} \leq_p \text{3SAT} \leq_p \text{4SAT} \leq_p \text{5SAT} \leq_p \ldots \leq_p \text{SAT}}$$

(a formula with 'at most N' literals trivially has 'at most $(N+1)$' literals)

‘Easy’ instances of SAT

- More surprisingly, we can find a reduction that goes the **other way**

$$\boxed{\text{SAT} \ \leq_p \ \ldots \ \leq_p \ \text{5SAT} \ \leq_p \ \text{4SAT} \ \leq_p \ \text{3SAT}}$$

- What is required?
  - 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!

‘Easy’ instances of SAT

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

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

‘Easy’ instances of SAT

Step 3) Introduce a fresh propositional variable to **‘bridge the gap’**

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

(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!

Q.E.D

‘Easy’ instances of SAT

- Why can’t we use this trick to reduce $\boxed{\text{SAT} \leq_p \text{2SAT}}$
  - Consider a single clause with 3 literals

$$\boxed{(\boldsymbol{P} \vee \neg \boldsymbol{Q} \vee \boldsymbol{R})}$$

  - We can break the clause into two pieces however we like,

$$\boxed{(\boldsymbol{P} \vee \neg \boldsymbol{Q} \quad \boldsymbol{R})}$$

  - However, by introducing a fresh variable, we run into problems!

$$\boxed{(\boldsymbol{P} \vee \neg \boldsymbol{Q} \vee \boldsymbol{X}) \ \wedge \ (\neg \boldsymbol{X} \vee \boldsymbol{R})}$$

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

- ...but maybe there is some **other trick** we could use?

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 clause as **two implications**

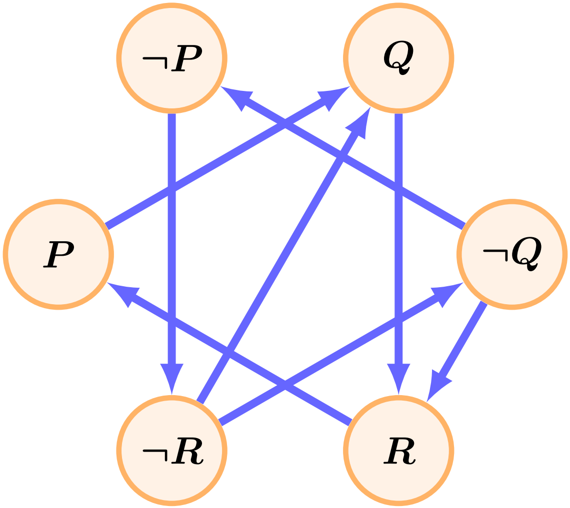
$$\boxed{(A \vee B) \quad \equiv \quad (\neg A \to B) \wedge (\neg B \to A)}$$

- **Example:**

$$\begin{array}{l} (\neg P \vee Q) \\ (\neg Q \vee R) \\ (P \vee \neg R) \\ (R \vee Q) \end{array} \quad \Rightarrow \quad \begin{array}{lcl} (P \to Q) & \wedge & (\neg Q \to \neg P) \\ (Q \to R) & \wedge & (\neg R \to \neg Q) \\ (\neg P \to \neg R) & \wedge & (R \to P) \\ (\neg R \to Q) & \wedge & (\neg Q \to R) \end{array}$$

Solving the 2SAT Problem

Step 2) Construct the **implication graph**



$(P \to Q)$

$(\neg Q \to \neg P)$

$(Q \to R)$

$(\neg R \to \neg Q)$

$(\neg P \to \neg R)$

$(R \to P)$

$(\neg R \to Q)$

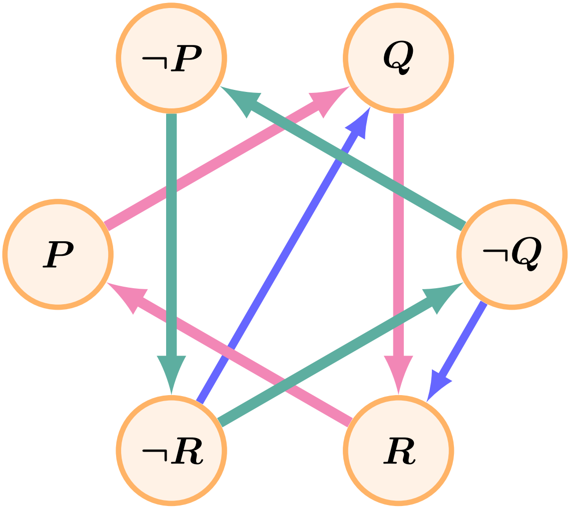
$(\neg Q \to R)$

Step 3) F is satisfiable iff every **strongly connected component** is consistent.

(consistent = does not contain a literal and its negation)

Solving the 2SAT Problem

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)

Solving the 2SAT Problem

Conclusion) The reduction 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!)

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

Q.E.D

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 decided in polynomial time! **(unless P $=$ NP ... we don't know!)**

- Is **every formula** with 3 literals per clause is hard to solve?

- **Propositional Horn Clauses:**

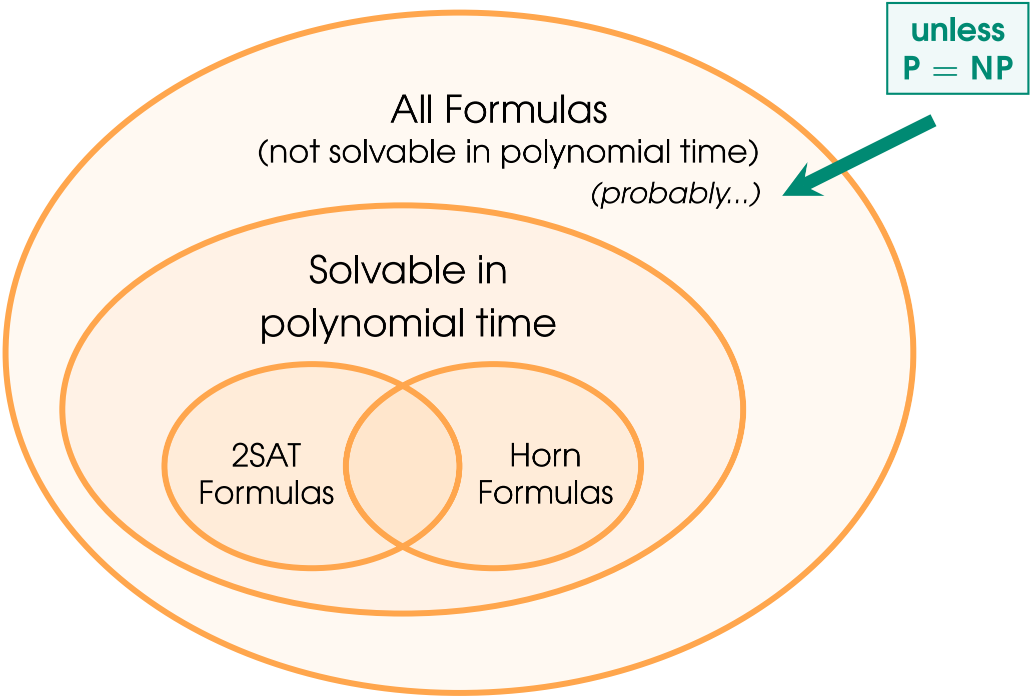
$$\boxed{P,\ (\neg P \lor Q),\ (\neg P \lor \neg Q \lor R),\ (\neg Q \lor S),\ (\neg S \lor \neg R \lor T)}$$

(each clause contains at most one positive literal)

Can be solved with just Unit Propogation and Pure Literal Elimination

Summary

- **Breakdown of all (propositional) formulas:**



Next Time...

- Heuristic Search
- Optimisation Algorithms
    - Revisiting Travelling Salesman Problem

End of Slides!

🎉

Feedback

- **Let me know how you found today's lecture?**

`https://goo.gl/forms/5TVc7aDIEjZ5WyhC2`