

# Logic in AI

## Chapter 7

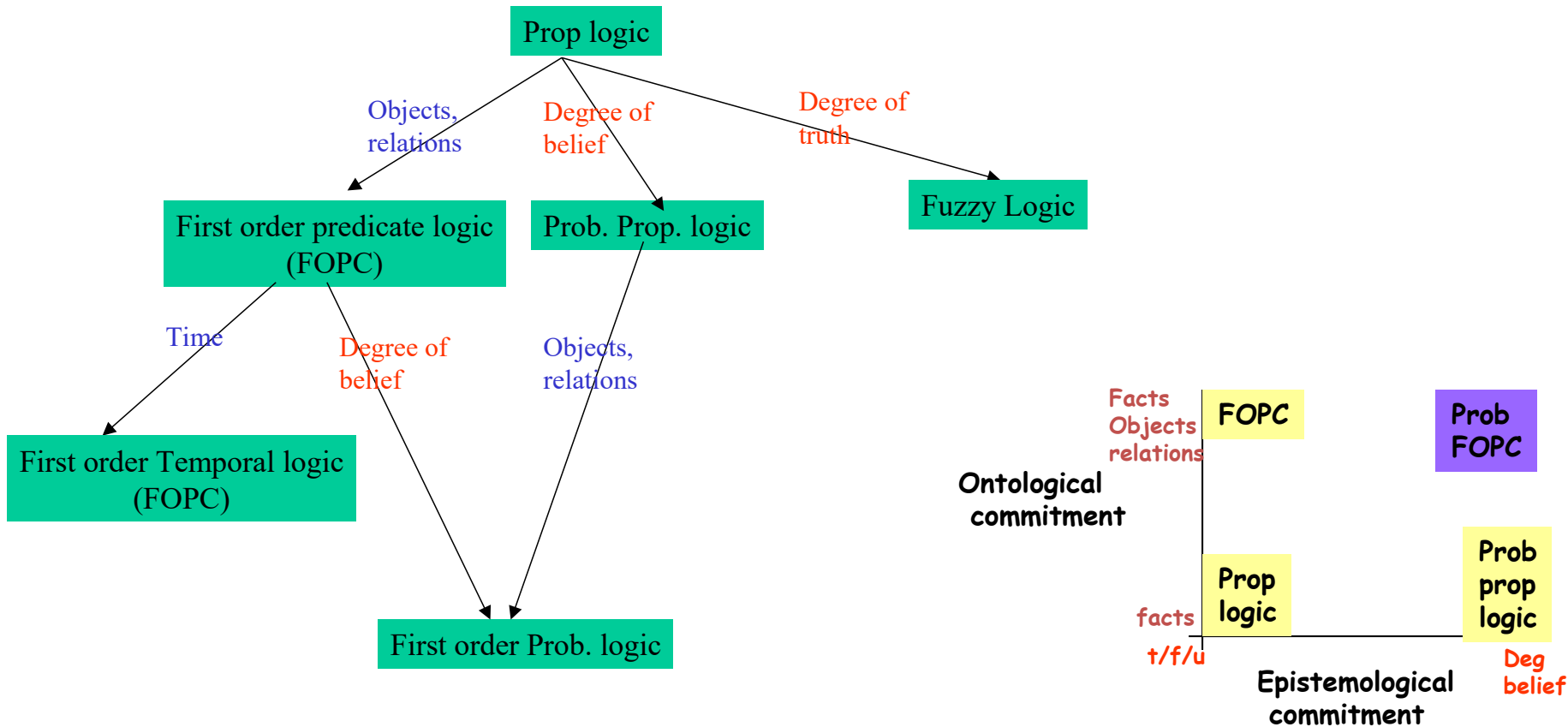
**Mausam**

(Based on slides of Dan Weld, Stuart Russell, Subbarao Kambhampati, Henry Kautz, Dieter Fox, and other UW AI Faculty)

# Knowledge Representation

- *represent knowledge about the world in a manner that facilitates inferencing (i.e. drawing conclusions) from knowledge.*
- Example: Arithmetic logic
  - $x \geq 5$
- In AI: typically based on
  - Logic
  - Probability
  - Logic and Probability

# Common KR Languages



# KR Languages

- Propositional Logic
- Predicate Calculus
- Frame Systems
- Rules with Certainty Factors
- Bayesian Belief Networks
- Influence Diagrams
- Ontologies
- Semantic Networks
- Concept Description Languages
- Non-monotonic Logic

# Basic Idea of Logic

- By starting with true assumptions, you can deduce true conclusions.

# Truth

- Francis Bacon (1561-1626)

No pleasure is comparable to the standing upon the vantage-ground of truth.

- Thomas Henry Huxley (1825-1895)

Irrationally held truths may be more harmful than reasoned errors.

- John Keats (1795-1821)

Beauty is truth, truth beauty; that is all ye know on earth, and all ye need to know.

- Blaise Pascal (1623-1662)

We know the truth, not only by the reason, but also by the heart.

- François Rabelais (c. 1490-1553)

Speak the truth and shame the Devil.

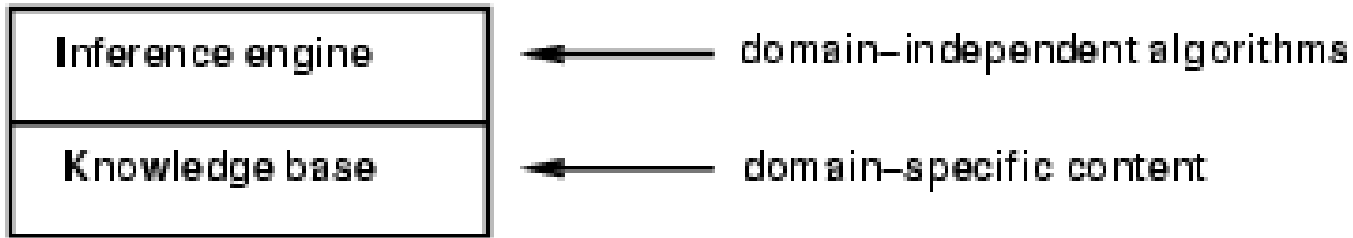
- Daniel Webster (1782-1852)

There is nothing so powerful as truth, and often nothing so strange.

# Components of KR

- Syntax: defines the sentences in the language
- Semantics: defines the “meaning” to sentences
- Inference Procedure
  - Algorithm
  - Sound?
  - Complete?
  - Complexity
- Knowledge Base

# Knowledge bases



- Knowledge base = set of **sentences** in a **formal** language
- **Declarative** approach to building an agent (or other system):
  - Tell it what it needs to know
- Then it can **Ask** itself what to do - answers should follow from the KB
- Agents can be viewed at the **knowledge level**  
i.e., what they know, regardless of how implemented
- Or at the **implementation level**  
i.e., data structures in KB and algorithms that manipulate them



# Propositional Logic

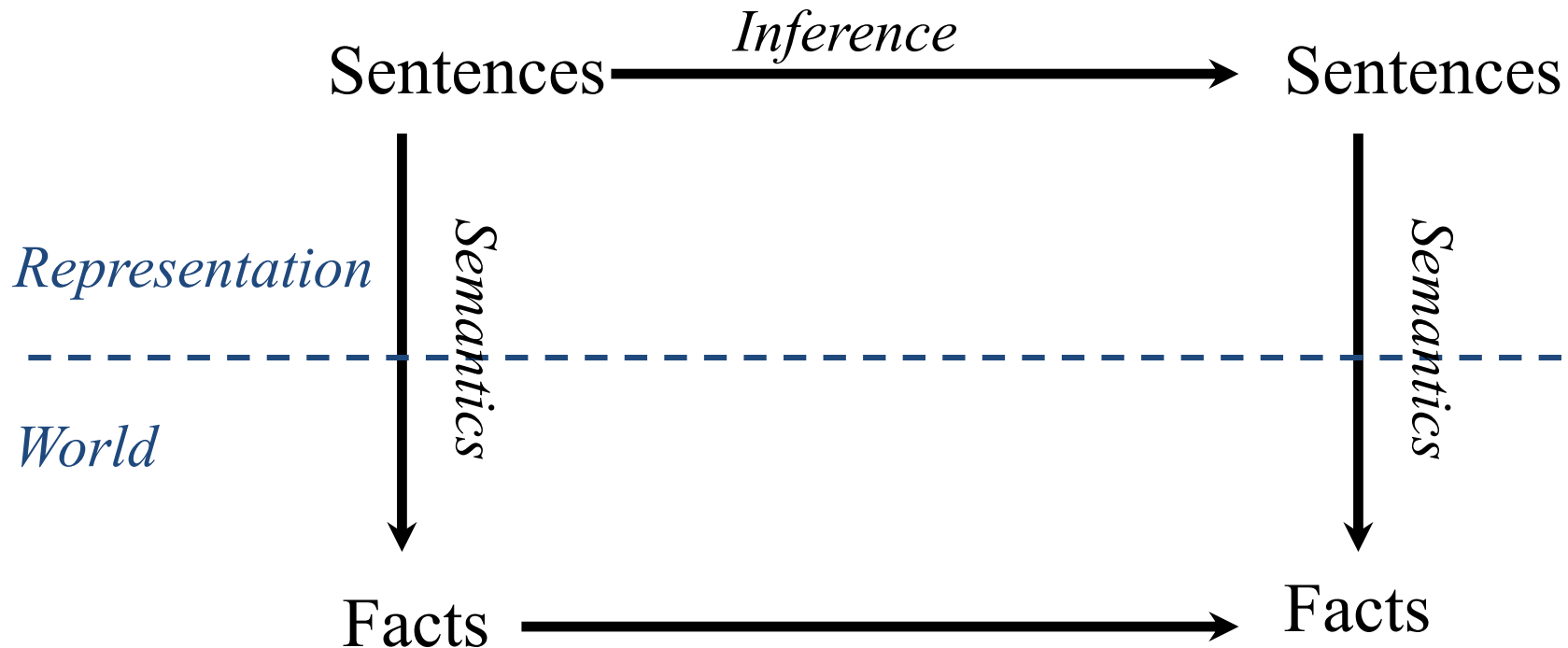
- Syntax
  - Atomic sentences:  $P, Q, \dots$
  - Connectives:  $\wedge, \vee, \neg, \rightarrow$
- Semantics
  - Truth Tables
- Inference
  - Modus Ponens
  - Resolution
  - DPLL
  - GSAT

# Propositional Logic: Syntax

- Atoms
  - $P, Q, R, \dots$
- Literals
  - $P, \neg P$
- Sentences
  - Any literal is a sentence
  - If  $S$  is a sentence
    - Then  $(S \wedge S)$  is a sentence
    - Then  $(S \vee S)$  is a sentence
- Conveniences
  - $P \rightarrow Q$  same as  $\neg P \vee Q$

# Semantics

- **Syntax**: which arrangements of symbols are *legal*
  - (Def “sentences”)
- **Semantics**: what the symbols *mean* in the world
  - (Mapping between symbols and worlds)



# Propositional Logic: SEMANTICS

- “Interpretation” (or “possible world”)
  - Assignment to each variable either T or F
  - Assignment of T or F to each connective via defns

		Q	
		T	F
P	T	<b>T</b>	<b>F</b>
	F	<b>F</b>	<b>F</b>

$P \wedge Q$

		Q	
		T	F
P	T	<b>T</b>	<b>T</b>
	F	<b>T</b>	<b>F</b>

$P \vee Q$

# Satisfiability, Validity, & Entailment

- S is **satisfiable** if it is true in *some* world
- S is **unsatisfiable** if it is false in *all* worlds
- S is **valid** if it is true in *all* worlds
- S1 **entails** S2 if *wherever* S1 is true S2 is also true

# Examples

$$P \rightarrow Q$$

$$R \rightarrow \neg R$$

$$S \wedge (W \wedge \neg S)$$

$$T \vee \neg T$$

$$X \rightarrow X$$

# Notation

$\Rightarrow$

$\supset$

$\rightarrow$

$\vdash$

$\models$



**Implication** (syntactic symbol)

**Proves:**  $S1 \vdash_{ie} S2$  if 'ie' algorithm says 'S2' from S1

**Entails:**  $S1 \models S2$  if wherever S1 is true S2 is also true

- **Sound**

$\vdash \rightarrow \models$

- **Complete**

$\models \rightarrow \vdash$

- *(all truth & nothing but the truth)*

# Reasoning Tasks

- **Model finding**

KB = background knowledge

S = description of problem

Show  $(KB \wedge S)$  is satisfiable

A kind of **constraint satisfaction**

- **Deduction**

S = question

Prove that  $KB \models S$

Two approaches:

- **Rules to derive new formulas from old (inference)**
- **Show  $(KB \wedge \neg S)$  is unsatisfiable**



# Special Syntactic Forms

- General Form:

$$((q \wedge \neg r) \rightarrow s) \wedge \neg (s \wedge t)$$

- Conjunction Normal Form (CNF)

$$(\neg q \vee r \vee s) \wedge (\neg s \vee \neg t)$$

Set notation:  $\{ (\neg q, r, s), (\neg s, \neg t) \}$

empty clause  $() = \text{false}$

- Binary clauses: 1 or 2 literals per clause

$$(\neg q \vee r) \qquad (\neg s \vee \neg t)$$

- Horn clauses: 0 or 1 positive literal per clause

$$(\neg q \vee \neg r \vee s) \qquad (\neg s \vee \neg t)$$

$$(q \wedge r) \rightarrow s \qquad (s \wedge t) \rightarrow \text{false}$$

# Propositional Logic: Inference

A *mechanical* process for computing new sentences

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. SAT
  1. Davis Putnam
  2. WalkSat

# Inference 1: Forward Chaining

## Forward Chaining

Based on rule of *modus ponens*

If know  $P_1, \dots, P_n$  & know  $(P_1 \wedge \dots \wedge P_n) \rightarrow Q$

Then can conclude  $Q$

## Backward Chaining: search

start from the query and go backwards

# Analysis

- Sound?
- Complete?

Can you prove  
 $\{ \} \models Q \vee \neg Q$

- If KB has only Horn clauses & query is a single literal
  - Forward Chaining is complete
  - Runs linear in the size of the KB

# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

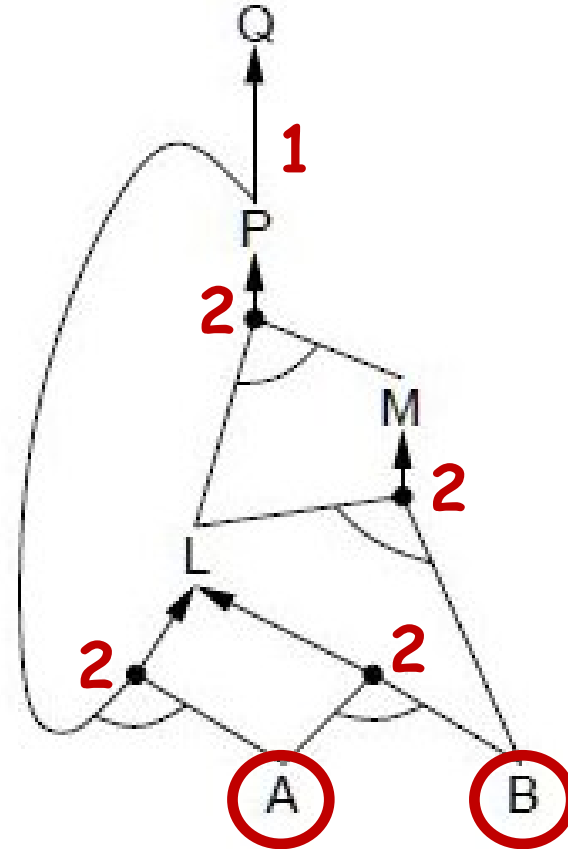
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

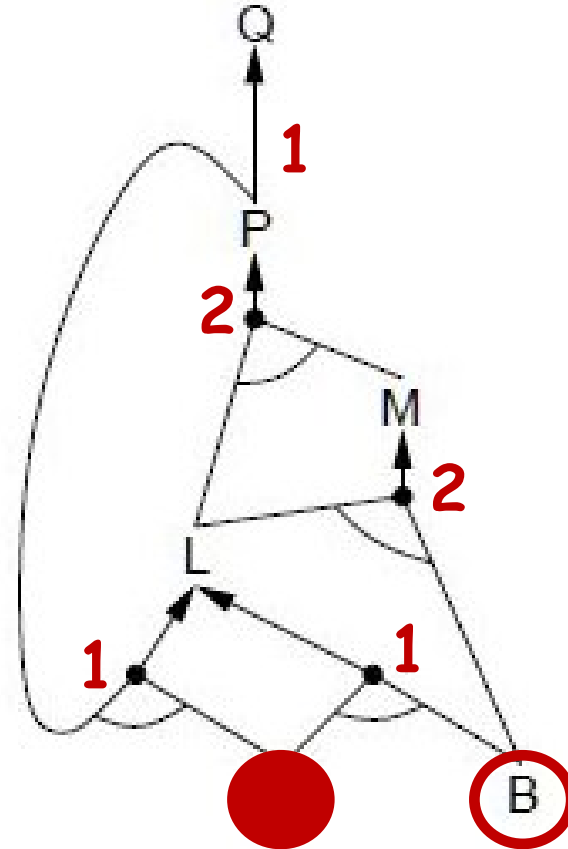
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

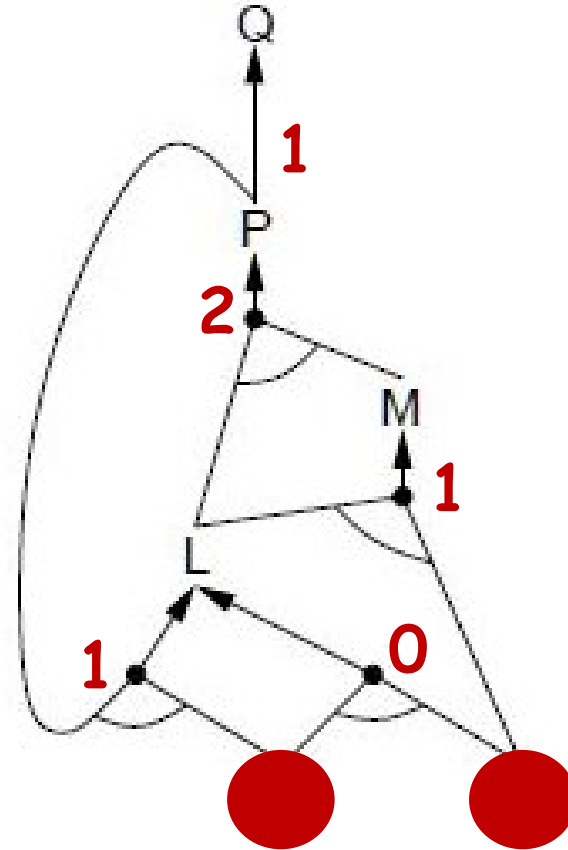
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

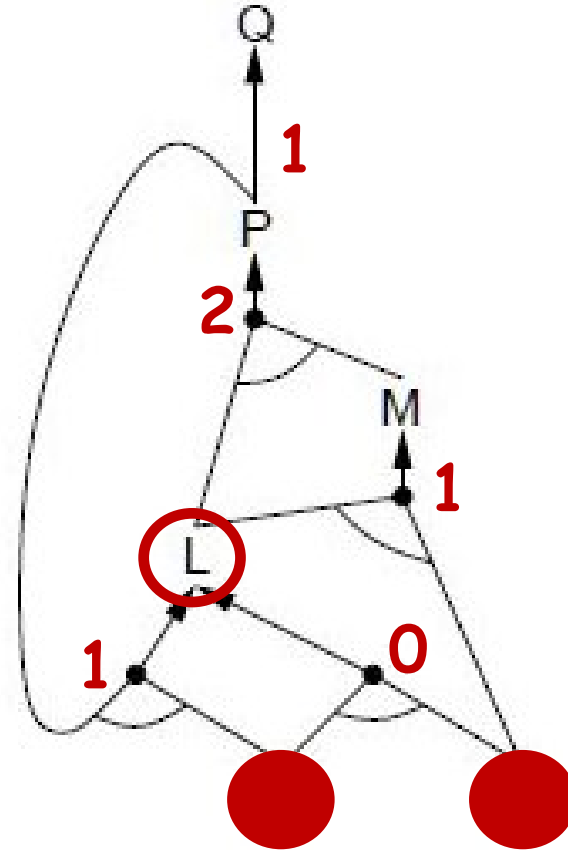
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$





# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

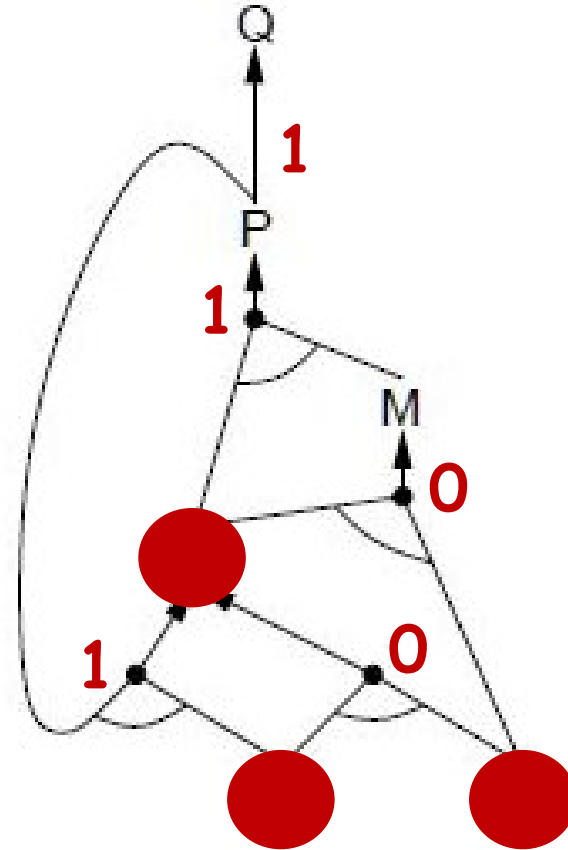
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

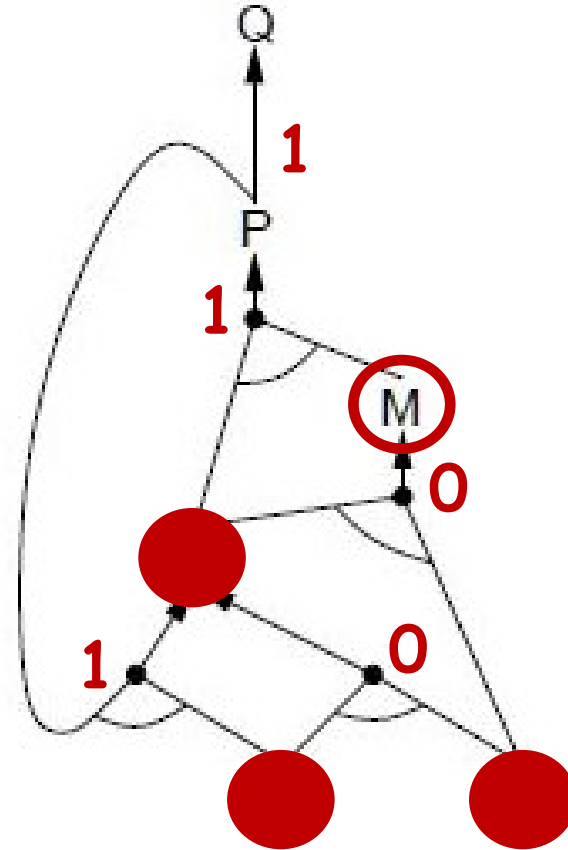
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

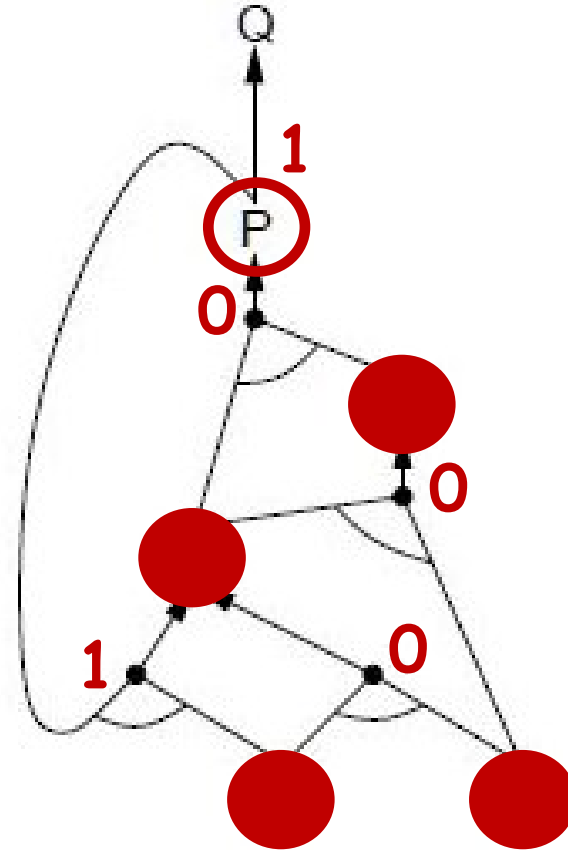
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

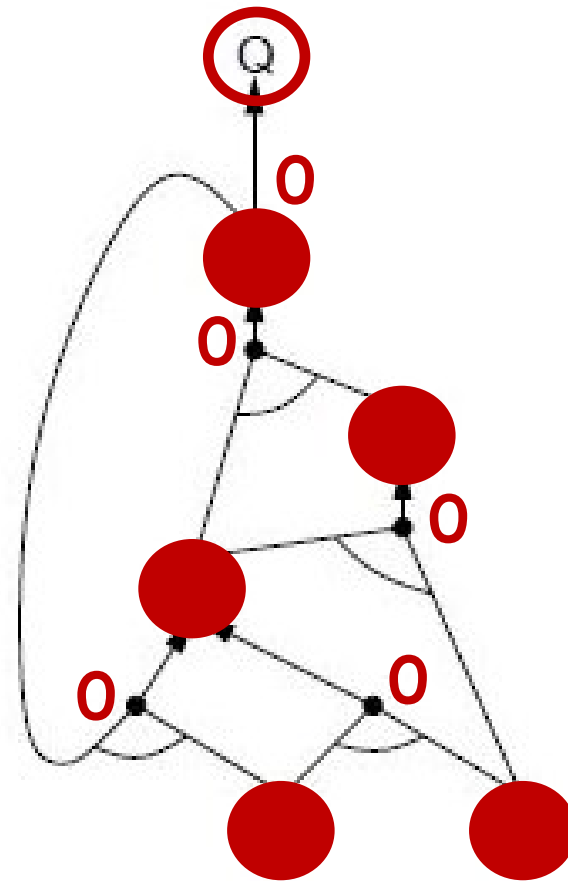
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Example

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

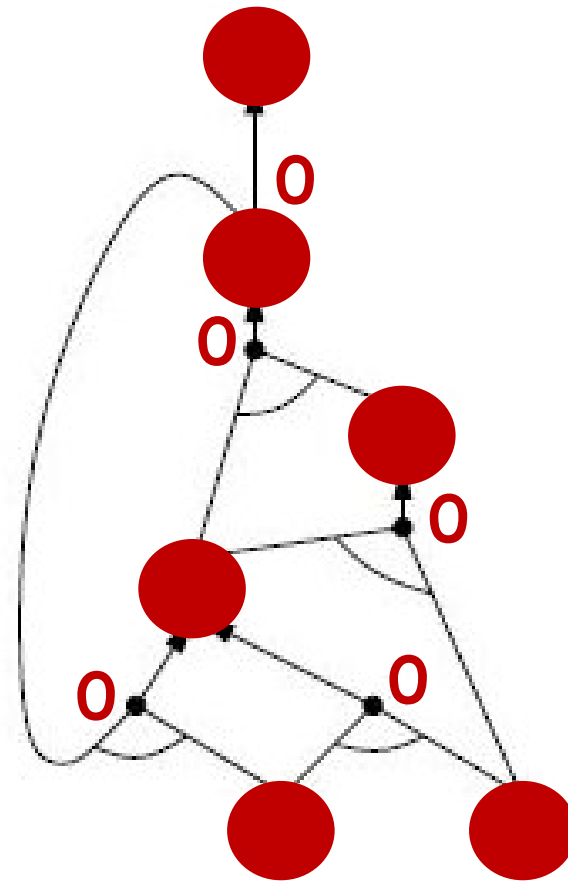
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$A$

$B$



# Propositional Logic: Inference

A *mechanical* process for computing new sentences

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. SAT
  1. Davis Putnam
  2. WalkSAT

# Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Eliminate  $\Leftrightarrow$ , replacing  $\alpha \Leftrightarrow \beta$  with  $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ .

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminate  $\Rightarrow$ , replacing  $\alpha \Rightarrow \beta$  with  $\neg\alpha \vee \beta$ .

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. Move  $\neg$  inwards using de Morgan's rules and double-negation:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Apply distributivity law ( $\vee$  over  $\wedge$ ) and flatten:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

# Inference 2: Resolution

[Robinson 1965]

$$\{ (p \vee \alpha), (\neg p \vee \beta \vee \gamma) \} \vdash_{\text{R}} (\alpha \vee \beta \vee \gamma)$$

Correctness

If  $S1 \vdash_{\text{R}} S2$  then  $S1 \models S2$

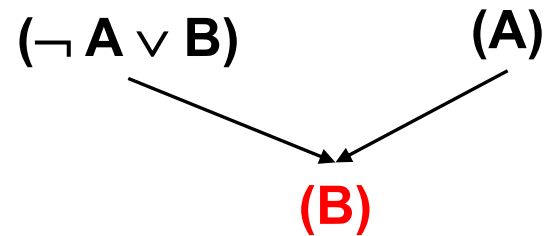
Refutation Completeness:

If  $S$  is unsatisfiable then  $S \vdash_{\text{R}} ()$



# Resolution subsumes Modus Ponens

$$A \rightarrow B, A \models B$$



If Will goes, Jane will go

$\sim W \vee J$

If doesn't go, Jane will still go

$W \vee J$

Will Jane go?

$\models J?$

$J \vee J = J$

Don't need to use other  
equivalences if we use  
resolution in *refutation* style

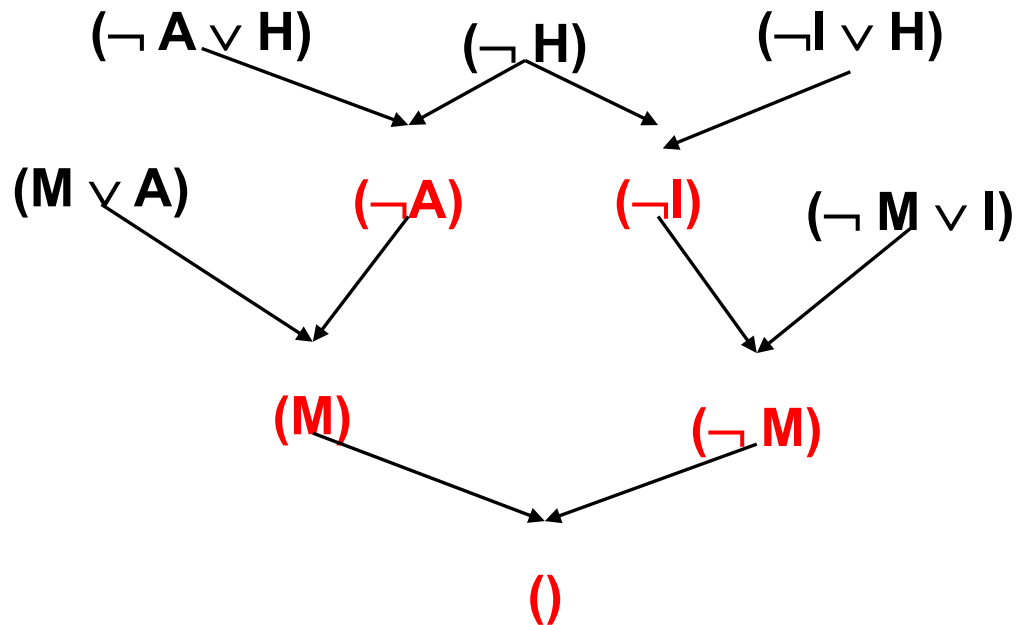
$\sim J$   
 $\sim W \vee J$   
 $W \vee J$   
 $\sim W$   
 $J$   
 $\square$

# Resolution

*If the unicorn is mythical, then it is immortal, but if it is not mythical, it is a mammal. If the unicorn is either immortal or a mammal, then it is horned.*

**Prove: the unicorn is horned.**

**M** = mythical  
**I** = immortal  
**A** = mammal  
**H** = horned



# Search in Resolution

- Convert the database into clausal form  $D_c$
- Negate the goal first, and *then* convert it into clausal form  $D_G$
- Let  $D = D_c + D_G$
- Loop
  - Select a pair of Clauses C1 and C2 from D
    - Different control strategies can be used to select C1 and C2 to reduce number of resolutions tries
  - Resolve C1 and C2 to get C12
  - If C12 is empty clause, QED!! Return Success (We proved the theorem; )
  - $D = D + C12$
- Out of loop but no empty clause. Return “Failure”
  - Finiteness is guaranteed if we make sure that:
    - we never resolve the same pair of clauses more than once;
    - we use factoring, which removes multiple copies of literals from a clause (e.g. QVPVP  $\Rightarrow$  QVP)

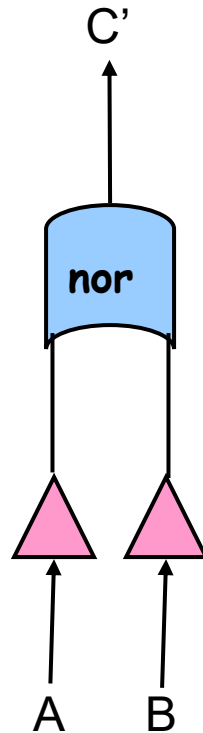
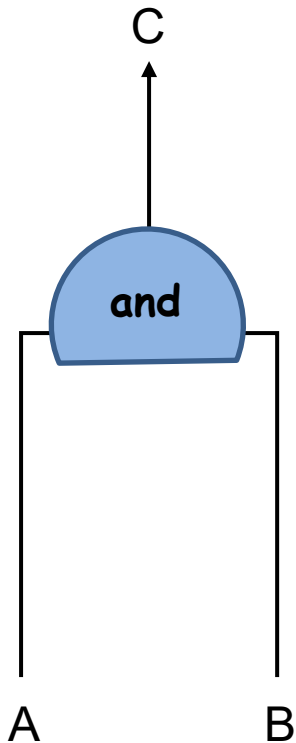
# SAT: Model Finding

- Find assignments to variables that makes a formula true

# Why study Satisfiability?

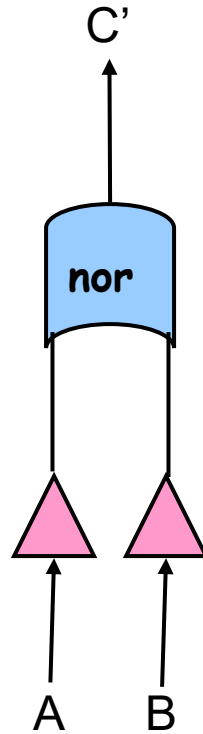
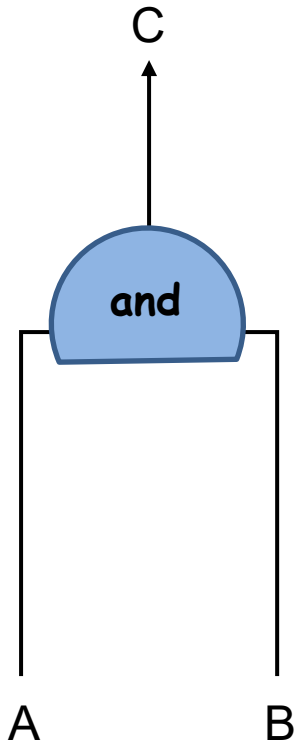
- Canonical NP complete problem.
  - several hard problems modeled as SAT
- Tonne of applications
- State-of-the-art solvers superfast

# Testing Circuit Equivalence



- Do two circuits compute the same function?
- Circuit optimization
- Is there input for which the two circuits compute different values?

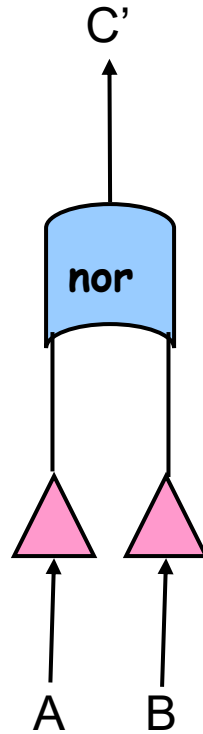
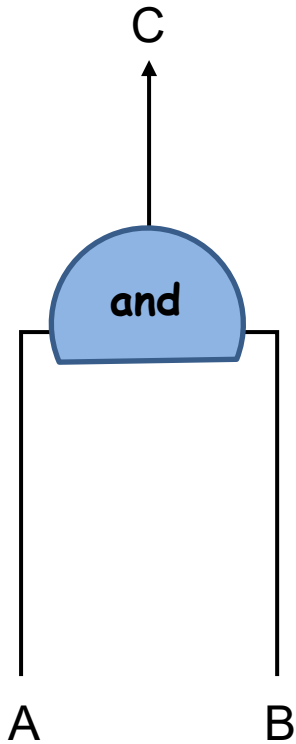
# Testing Circuit Equivalence



$$\begin{aligned}C &\equiv A \wedge B \\C' &\equiv \neg(D \vee E) \\D &\equiv \neg A \\E &\equiv \neg B\end{aligned}$$



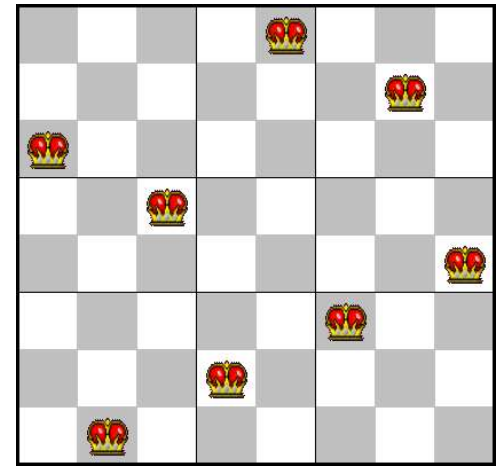
# Testing Circuit Equivalence



$$\begin{aligned}C &\equiv A \wedge B \\C' &\equiv \neg(D \vee E) \\D &\equiv \neg A \\E &\equiv \neg B \\&\neg(C \equiv C')\end{aligned}$$

# SAT Translation of N-Queens

- At least one queen each column:  
 $(Q_{11} \vee Q_{12} \vee Q_{13} \vee \dots \vee Q_{18})$   
 $(Q_{21} \vee Q_{22} \vee Q_{23} \vee \dots \vee Q_{28})$   
...
- No attacks:  
 $(\sim Q_{11} \vee \sim Q_{12})$   
 $(\sim Q_{11} \vee \sim Q_{22})$   
 $(\sim Q_{11} \vee \sim Q_{21})$   
...



# Graph Coloring

- A new SAT Variable for var-val pair

$X_{WA-r}, X_{WA-g}, X_{WA-b}, X_{NT-r} \dots$

- Each var has at least 1 value

–  $X_{WA-r} \vee X_{WA-g} \vee X_{WA-b}$

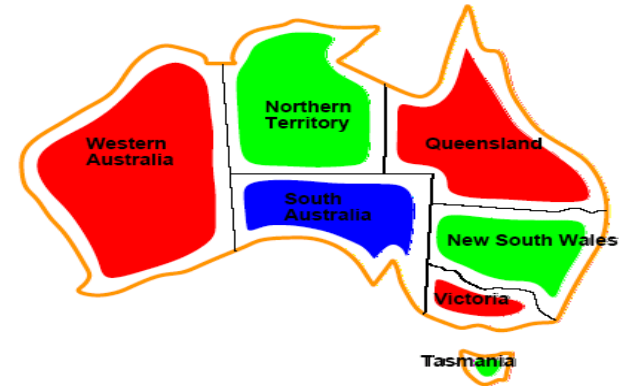
- No var has two values

–  $\sim X_{WA-r} \vee \sim X_{WA-g}$

–  $\sim X_{WA-r} \vee \sim X_{WA-b}$

- Constraints

–  $\sim X_{WA-r} \vee \sim X_{NT-r}$



# Application: Diagnosis

- Problem: diagnosis a malfunctioning device
  - Car
  - Computer system
  - Spacecraft
- where
  - Design of the device is known
  - We can observe the state of only certain parts of the device – much is hidden

# Model-Based, Consistency-Based Diagnosis

- Idea: create a logical formula that describes how the device should work
  - Associated with each “breakable” component C is a proposition that states “C is okay”
  - Sub-formulas about component C are all conditioned on C being okay
- A diagnosis is a smallest of “not okay” assumptions that are consistent with what is actually observed

# Consistency-Based Diagnosis

1. Make some **Observations** O.
2. Initialize the **Assumption Set** A to assert that all components are working properly.
3. Check if the KB, A, O together are **inconsistent** (can deduce *false*).
4. If so, delete propositions from A until **consistency is restored** (cannot deduce *false*).  
The deleted propositions are a diagnosis.

*There may be many possible diagnoses*

# Example: Automobile Diagnosis

- *Observable Propositions:*  
EngineRuns, GasInTank, ClockRuns
- *Assumable Propositions:*  
FuelLineOK, BatteryOK, CablesOK, ClockOK
- *Hidden (non-Assumable) Propositions:*  
GasInEngine, PowerToPlugs
- *Device Description F:*  
 $(\text{GasInTank} \wedge \text{FuelLineOK}) \rightarrow \text{GasInEngine}$   
 $(\text{GasInEngine} \wedge \text{PowerToPlugs}) \rightarrow \text{EngineRuns}$   
 $(\text{BatteryOK} \wedge \text{CablesOK}) \rightarrow \text{PowerToPlugs}$   
 $(\text{BatteryOK} \wedge \text{ClockOK}) \rightarrow \text{ClockRuns}$
- *Observations:*  
 $\neg \text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}$

# Example

- Is  $F \cup \text{Observations} \cup \text{Assumptions}$  consistent?
- $F \cup \{\neg \text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}\}$   
 $\cup \{\text{FuelLineOK}, \text{BatteryOK}, \text{CablesOK}, \text{ClockOK}\} \rightarrow \text{false}$ 
  - *Must restore consistency!*
- $F \cup \{\neg \text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}\}$   
 $\cup \{\text{BatteryOK}, \text{CablesOK}, \text{ClockOK}\} \rightarrow \text{false}$ 
  - $\neg \text{FuelLineOK}$  is a diagnosis
- $F \cup \{\neg \text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}\}$   
 $\cup \{\text{FuelLineOK}, \text{CablesOK}, \text{ClockOK}\} \rightarrow \text{false}$ 
  - $\neg \text{BatteryOK}$  is not a diagnosis



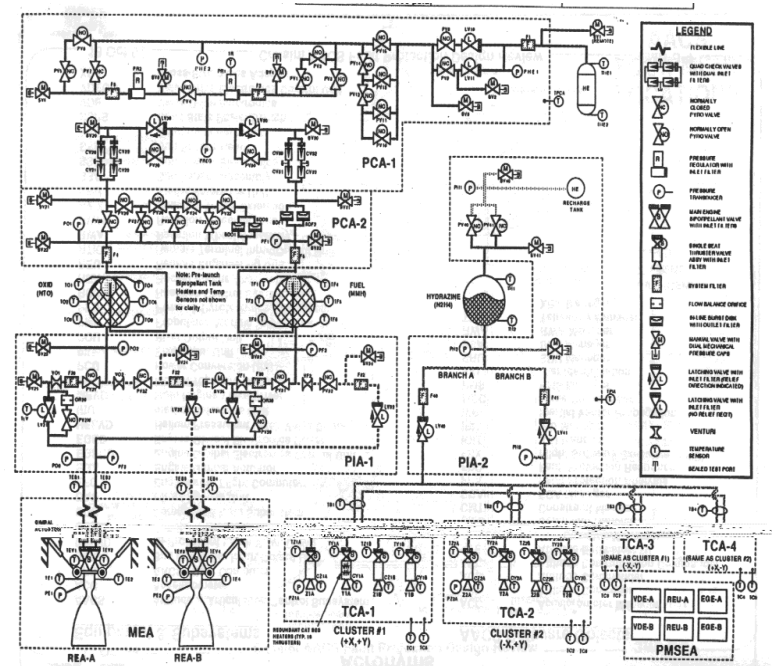
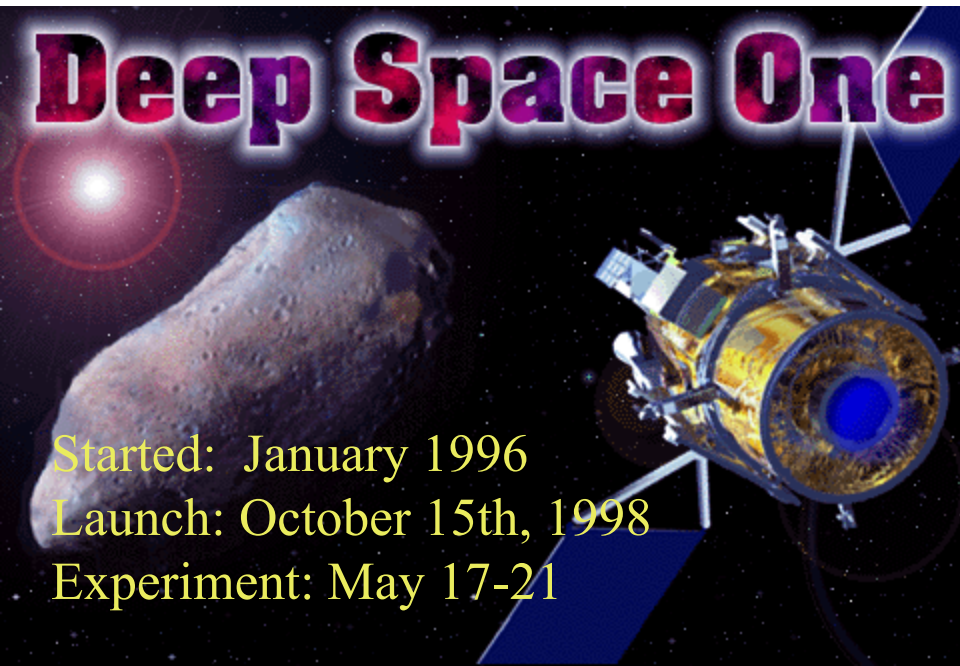
# Complexity of Diagnosis

- If  $F$  is **Horn**, then each consistency test takes linear time – unit propagation is complete for Horn clauses.
- Complexity = ways to delete propositions from Assumption Set that are considered.
  - Single fault diagnosis –  $O(n^2)$
  - Double fault diagnosis –  $O(n^3)$
  - Triple fault diagnosis –  $O(n^4)$

...

# Deep Space One

- Autonomous diagnosis & repair “Remote Agent”
- Compiled systems schematic to 7,000 var SAT problem



# Deep Space One

- a failed electronics unit
  - Remote Agent fixed by reactivating the unit.
- a failed sensor providing false information
  - Remote Agent recognized as unreliable and therefore correctly ignored.
- an attitude control thruster (a small engine for controlling the spacecraft's orientation) stuck in the "off" position
  - Remote Agent detected and compensated for by switching to a mode that did not rely on that thruster.

# Inference 3: Model Enumeration

```
for (m in truth assignments) {  
    if (m makes  $\Phi$  true)  
        then return "Sat!"  
}  
return "Unsat!"
```

# Inference 4: DPLL

## (Enumeration of *Partial* Models)

[Davis, Putnam, Loveland & Logemann 1962]

*Version 1*

```
dp11_1(pa) {  
  if (pa makes F false) return false;  
  if (pa makes F true) return true;  
  choose P in F;  
  if (dp11_1(pa  $\cup$  {P=0})) return true;  
  return dp11_1(pa  $\cup$  {P=1});  
}
```

Returns true if F is satisfiable, false otherwise

# DPLL Version 1

$$(a \vee b \vee c)$$

$$(a \vee \neg b)$$

$$(a \vee \neg c)$$

$$(\neg a \vee c)$$

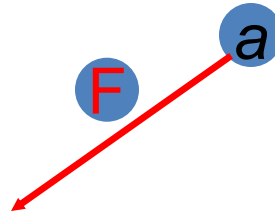
# DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



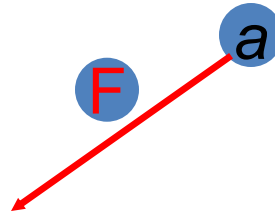
# DPLL Version 1

$(F \vee b \vee c)$

$(F \vee \neg b)$

$(F \vee \neg c)$

$(T \vee c)$





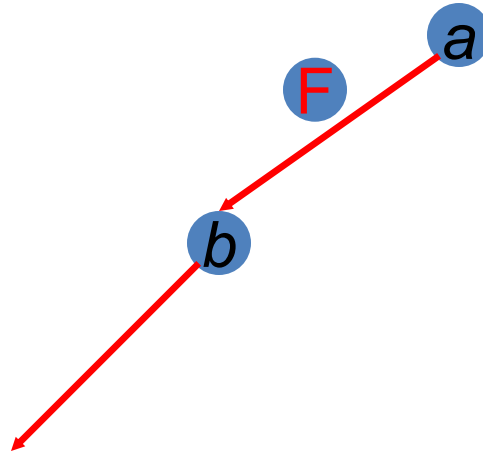
# DPLL Version 1

$(F \vee F \vee c)$

$(F \vee T)$

$(F \vee \neg c)$

$(T \vee c)$



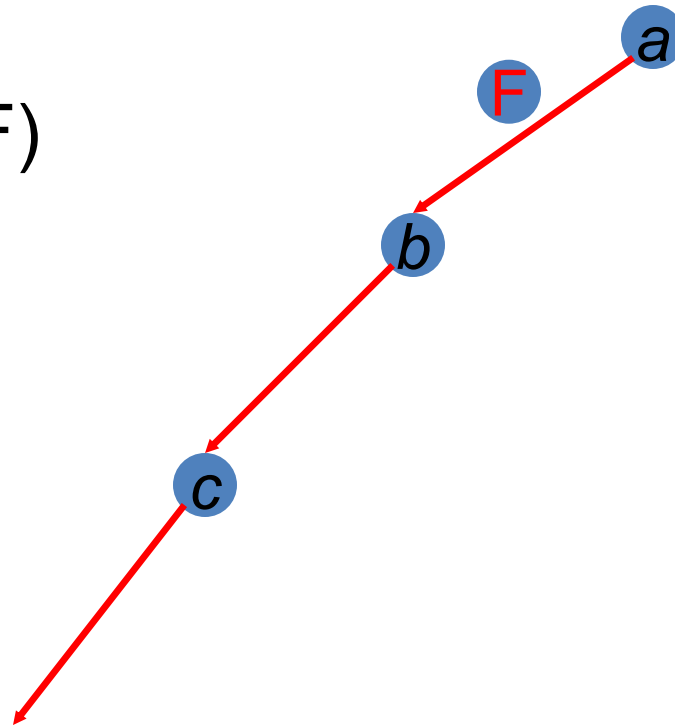
# DPLL Version 1

$(F \vee F \vee F)$

$(F \vee T)$

$(F \vee T)$

$(T \vee F)$



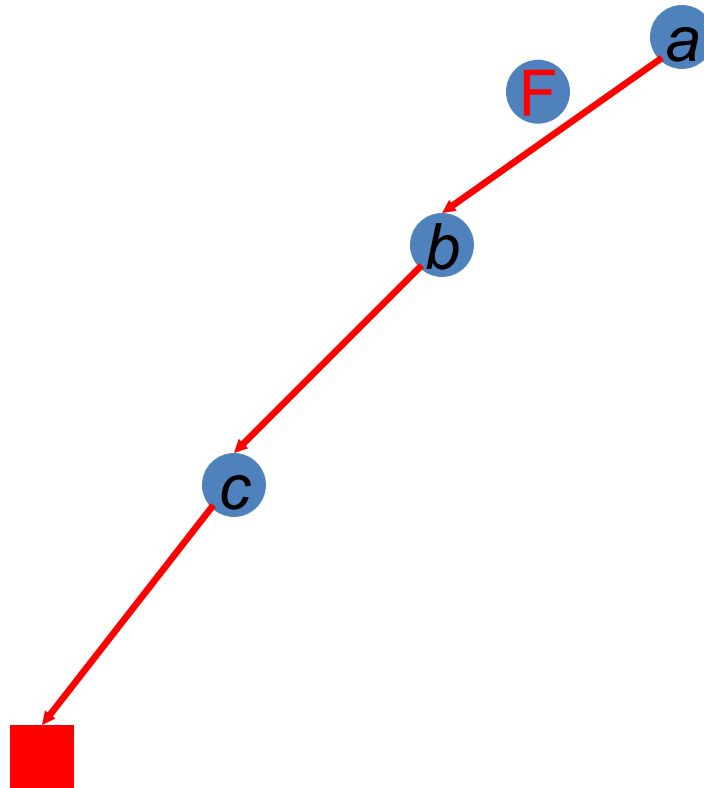
# DPLL Version 1

F

T

T

T



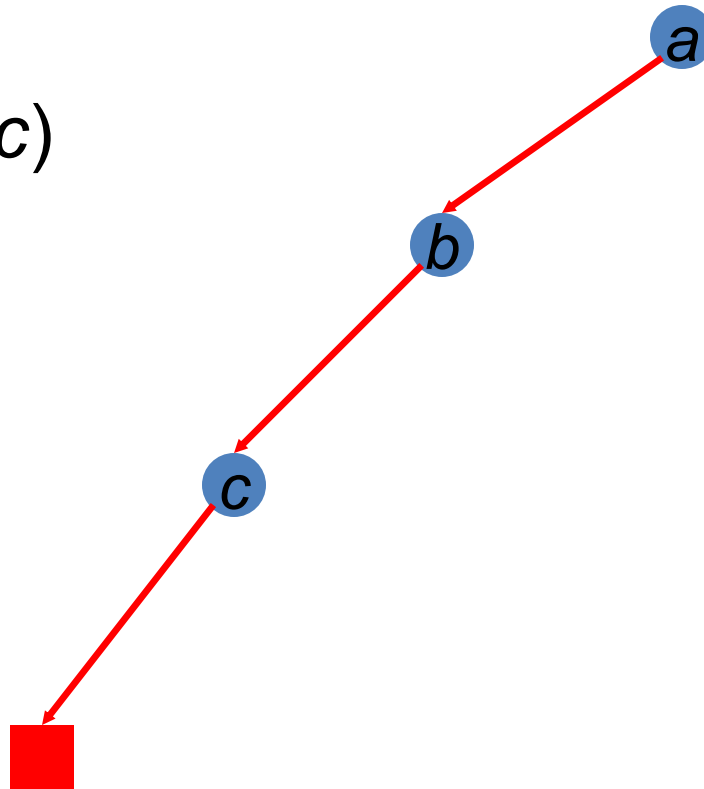
# DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



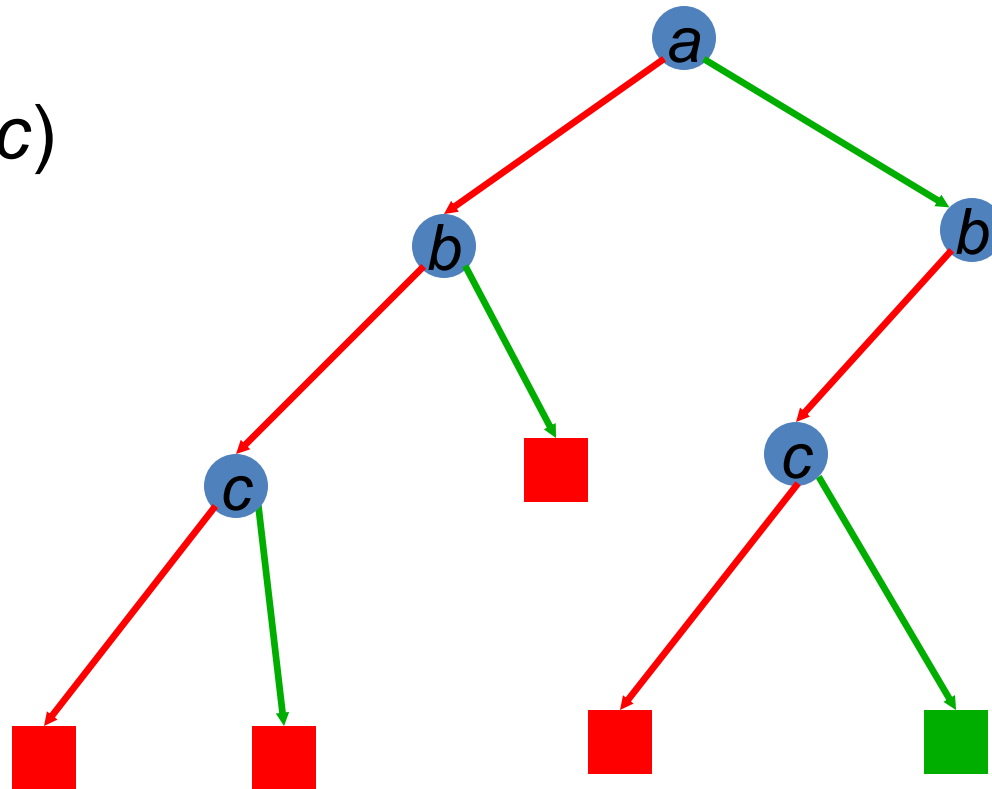
# DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



# DPLL as Search

- Search Space?
- Algorithm?

# Improving DPLL

If literal  $L_1$  is true, then clause  $(L_1 \vee L_2 \vee \dots)$  is true

If clause  $C_1$  is true, then  $C_1 \wedge C_2 \wedge C_3 \wedge \dots$  has the same value as  $C_2 \wedge C_3 \wedge \dots$

Therefore: Okay to delete clauses containing true literals!

If literal  $L_1$  is false, then clause  $(L_1 \vee L_2 \vee L_3 \vee \dots)$  has the same value as  $(L_2 \vee L_3 \vee \dots)$

Therefore: Okay to shorten clauses containing false literals!

If literal  $L_1$  is false, then clause  $(L_1)$  is false

Therefore: the empty clause means false!

## DPLL version 2

```
dpll_2(F, literal){  
    remove clauses containing literal  
    if (F contains no clauses)return true;  
    shorten clauses containing  $\neg$ literal  
    if (F contains empty clause)  
        return false;  
    choose V in F;  
    if (dpll_2(F,  $\neg$ V))return true;  
    return dpll_2(F, V);  
}
```



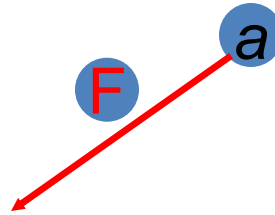
# DPLL Version 2

$(F \vee b \vee c)$

$(F \vee \neg b)$

$(F \vee \neg c)$

$(T \vee c)$

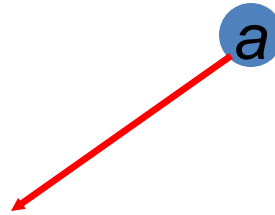


# DPLL Version 2

$(b \vee c)$

$(\neg b)$

$(\neg c)$

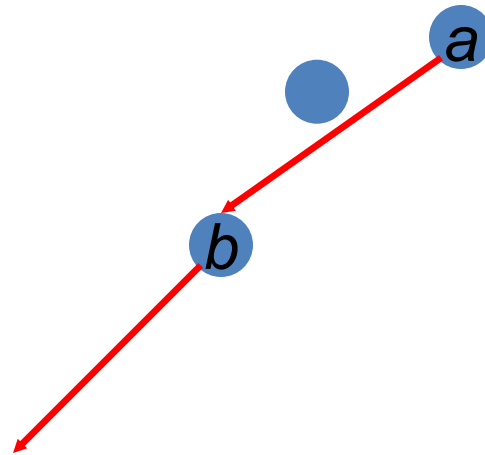


# DPLL Version 2

$(F \vee c)$

$(T)$

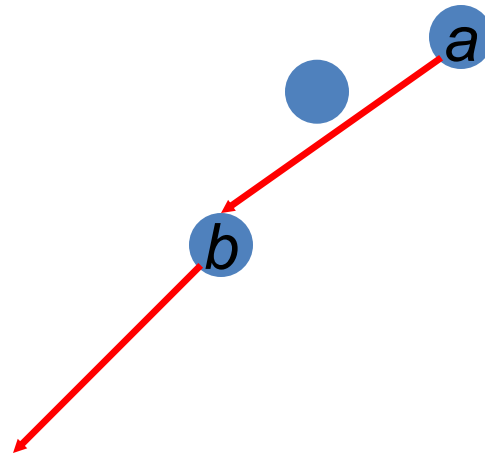
$(\neg c)$



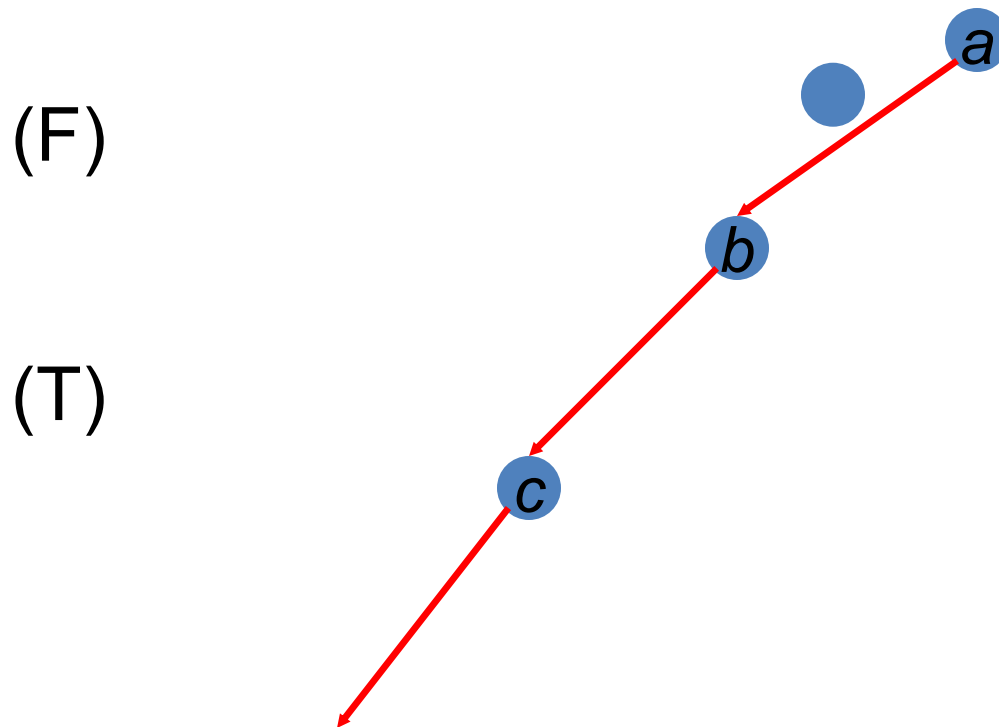
# DPLL Version 2

(c)

( $\neg c$ )

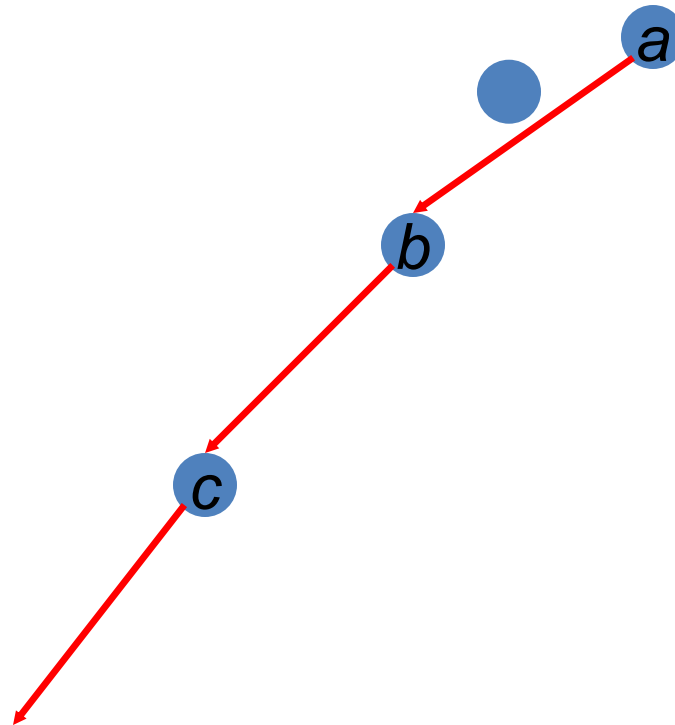


# DPLL Version 2



# DPLL Version 2

*Empty clause!*  
( )



# Structure in Clauses

- Unit Literals (unit propagation)

A literal that appears in a singleton clause

$\{\{\neg b \ c\}\{\neg c\}\{a \neg b \ e\}\{d \ b\}\{e \ a \ \neg c\}\}$

*Might as well set it true! And simplify*

$\{\{\neg b\}\{a \neg b \ e\}\{d \ b\}\}$   
 $\{\{d\}\}$

- Pure Literals

– A symbol that always appears with same sign

–  $\{\{a \neg b \ c\}\{\neg c \ d \ \neg e\}\{\neg a \neg b \ e\}\{d \ b\}\{e \ a \ \neg c\}\}$

*Might as well set it true! And simplify*

$\{\{a \neg b \ c\}\{\neg a \neg b \ e\}\{e \ a \ \neg c\}\}$

# In Other Words

Formula  $(L) \wedge C_2 \wedge C_3 \wedge \dots$  is only true when literal  $L$  is true

Therefore: Branch immediately on unit literals!

May view this as adding  
constraint propagation  
techniques into play



# In Other Words

Formula  $(L) \wedge C_2 \wedge C_3 \wedge \dots$  is only true when literal  $L$  is true

Therefore: Branch immediately on unit literals!

If literal  $L$  does not appear negated in formula  $F$ , then setting  $L$  true preserves satisfiability of  $F$

Therefore: Branch immediately on pure literals!

May view this as adding  
constraint propagation  
techniques into play

# DPLL (previous version)

Davis – Putnam – Loveland – Logemann

```
dp11(F, literal) {  
    remove clauses containing literal  
    if (F contains no clauses) return true;  
    shorten clauses containing  $\neg$ literal  
    if (F contains empty clause)  
        return false;  
  
    choose V in F;  
    if (dp11(F,  $\neg$ V)) return true;  
    return dp11(F, V);  
}
```

# DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dp11(F, literal) {  
    remove clauses containing literal  
    if (F contains no clauses) return true;  
    shorten clauses containing  $\neg$ literal  
    if (F contains empty clause)  
        return false;  
    if (F contains a unit or pure L)  
        return dp11(F, L);  
    choose V in F;  
    if (dp11(F,  $\neg$ V)) return true;  
    return dp11(F, V);  
}
```

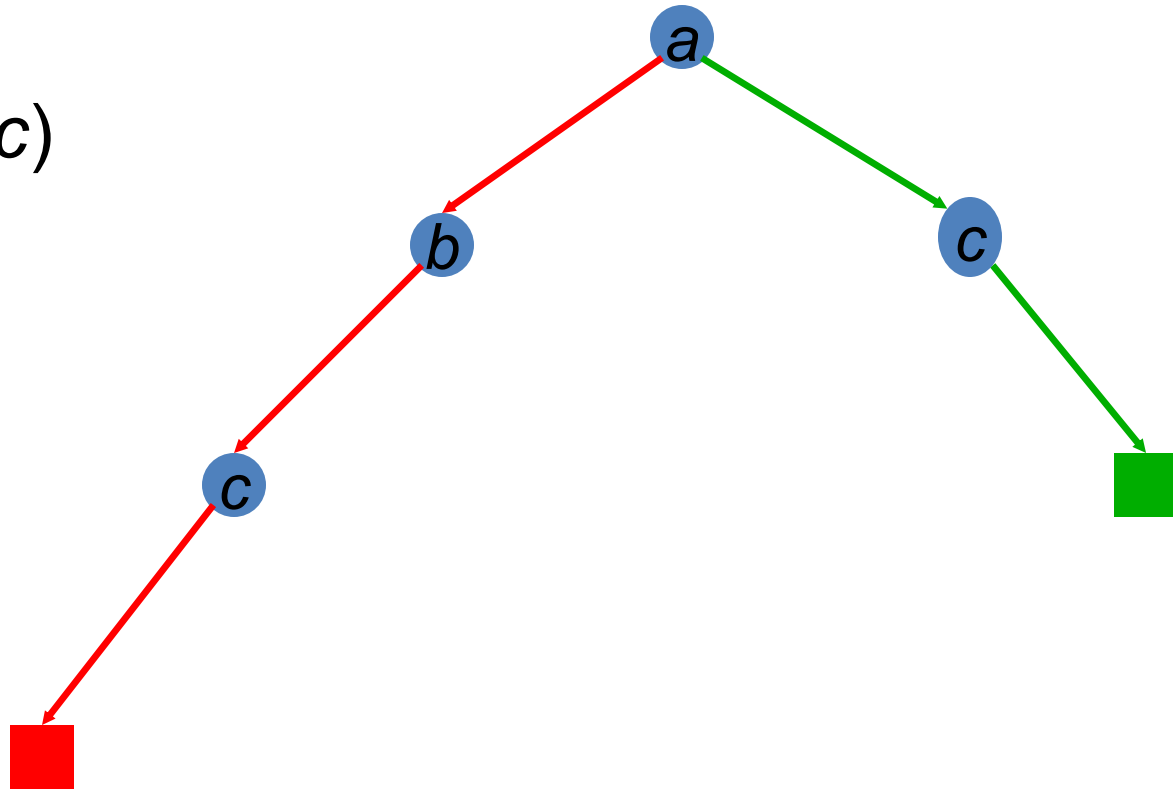
# DPLL (for real)

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



# DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){  
  remove clauses containing literal  
  if (F contains no clauses) return true;  
  shorten clauses containing  $\neg$ literal  
  if (F contains empty clause)  
    return false;  
  if (F contains a unit or pure L)  
    return dpll(F, L);  
  choose V in F;  
  if (dpll(F,  $\neg$ V)) return true;  
  return dpll(F, V);  
}
```

Where could we use a heuristic to further improve performance?

# Heuristic Search in DPLL

- Heuristics are used in DPLL to select a (non-unit, non-pure) proposition for branching
- Idea: identify a most constrained variable
  - Likely to create many unit clauses
- MOM's heuristic:
  - Most occurrences in clauses of minimum length

# GSAT

- **Local** search (Hill Climbing + Random Walk) over space of **complete** truth assignments
  - With prob  $p$ : flip **any** variable in any unsatisfied clause
  - With prob  $(1-p)$ : flip **best** variable in any unsat clause
    - best = one which minimizes #unsatisfied clauses
- SAT encodings of N-Queens, scheduling
- Best algorithm for random K-SAT
  - Best DPLL: 700 variables
  - Walksat: 100,000 variables

# Refining Greedy Random Walk

- Each flip
  - **makes** some false clauses become true
  - **breaks** some true clauses, that become false
- Suppose  $s1 \rightarrow s2$  by flipping  $x$ . Then:
$$\#unsat(s2) = \#unsat(s1) - make(s1,x) + break(s1,x)$$
- **Idea 1:** if a choice breaks nothing, it is very likely to be a good move
- **Idea 2:** near the solution, only the break count matters
  - the make count is usually 1



# Walksat

```
state = random truth assignment;
while ! GoalTest(state) do
  clause := random member { C | C is false in state };
  for each x in clause do compute break[x];
  if exists x with break[x]=0 then var := x;
  else
    with probability p do
      var := random member { x | x is in clause };
    else (probability 1-p)
      var := argminx { break[x] | x is in clause };
    endif
  state[var] := 1 - state[var];
end
return state;
```

**Put everything inside of a restart loop.  
Parameters:  $p$ ,  $\text{max\_flips}$ ,  $\text{max\_runs}$**

# Advantages of WalkSAT over GSAT

- WalkSat guaranteed to make at least 1 false clause true (in random walk also)
- Number of evaluations small per move
  - does not iterate over all variables
  - only variables in the sampled clause