Logic in AI

Chapter 7

Mausam

(Based on slides of Dan Weld, Stuart Russell, Subbarao Kambhampati, Dieter Fox, Henry Kautz…)

Logic: another thing that penguins aren't very good at.

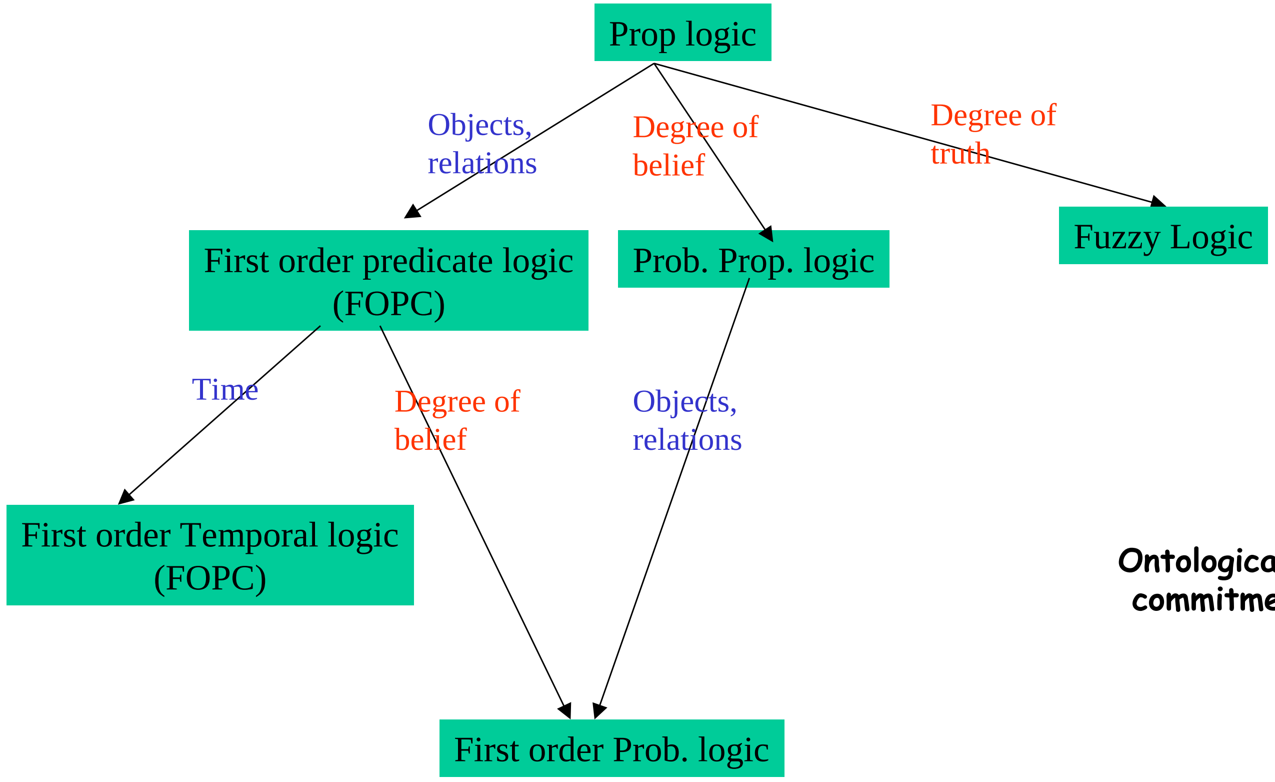
Knowledge Representation

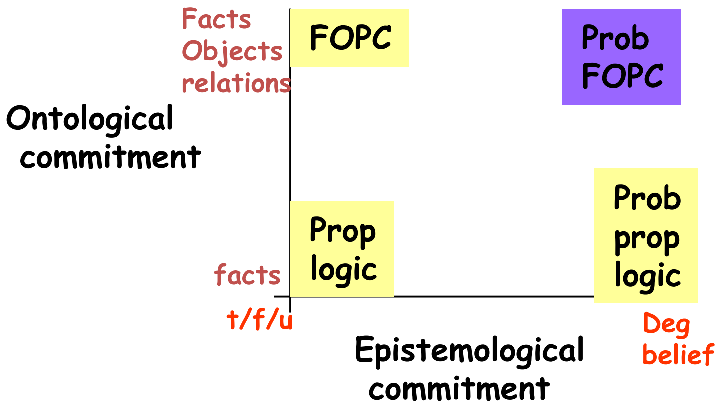
- *represent knowledge about the world in a manner that facilitates inferencing (i.e. drawing conclusions) from knowledge.*

- Example: Arithmetic logic
  - x >= 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.

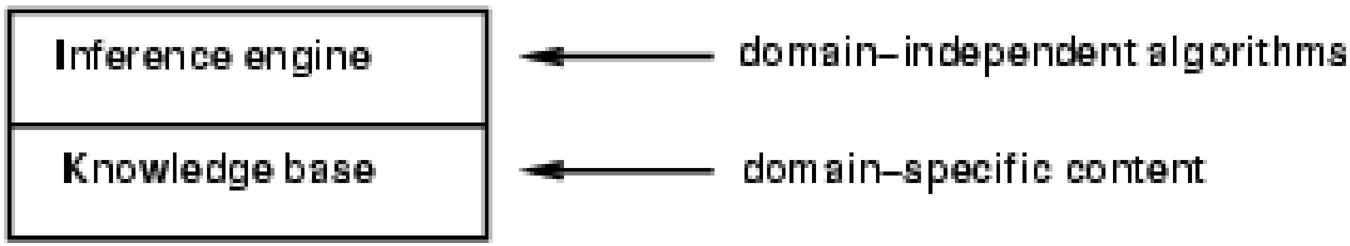
• Daniel Webster (1782-1852)

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, …
  - Connectives: $\wedge$, $\vee$, $\neg$, $\rightarrow$
- Semantics
  - Truth Tables
- Inference
  - Modus Ponens
  - Resolution
  - DPLL
  - GSAT

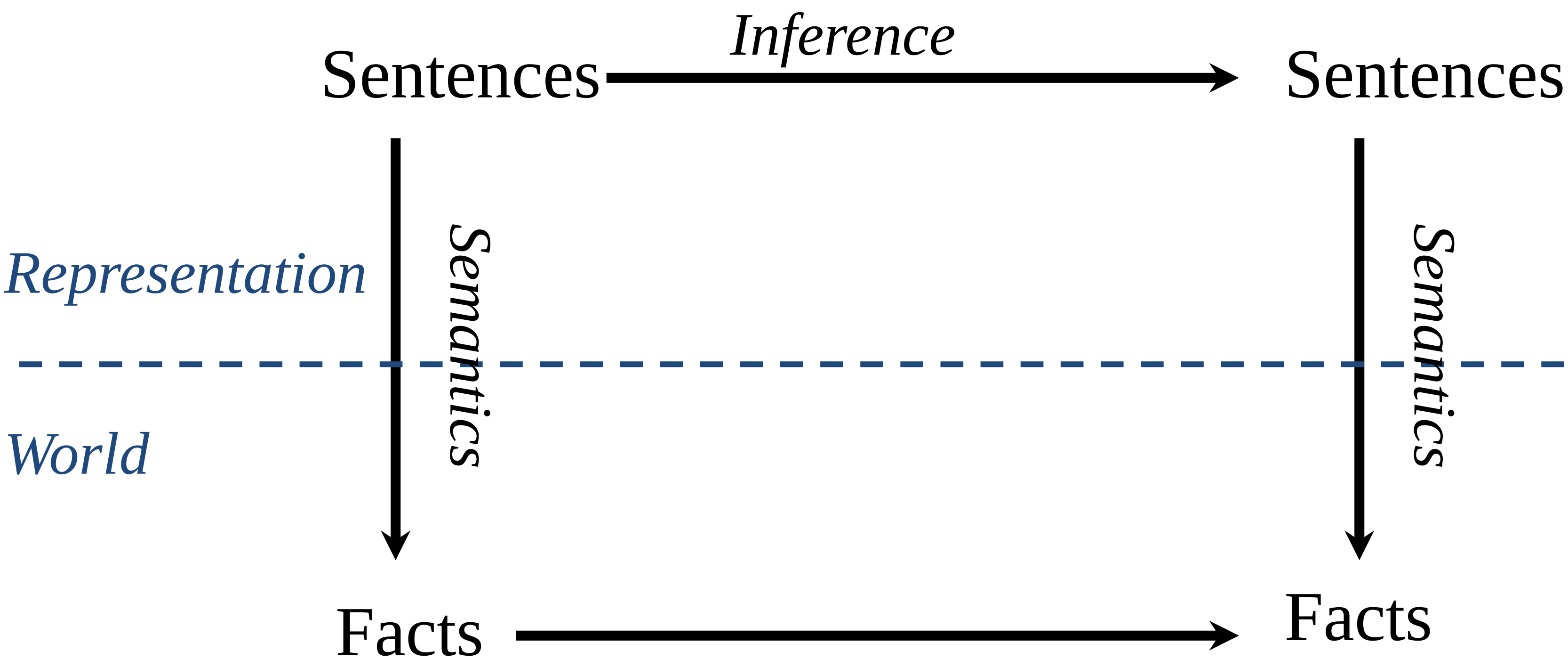
Propositional Logic: Syntax

- Atoms
  - P, Q, R, …
- 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

P \ Q	T	F
T	T	F
F	F	F

$P \wedge Q$

P \ Q	T	F
T	T	T
F	T	F

$P \vee Q$

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

$\Rightarrow$

$\supset$ } **Implication** (syntactic symbol)

$\rightarrow$

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

$\models$ **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 () = *false*

- Binary clauses: 1 or 2 literals per clause

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

- Horn clauses: 0 or 1 positive literal per clause

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

  $(q \wedge r) \rightarrow s$ $(s \wedge t) \rightarrow$ *false*

Propositional Logic: **Inference**

A ***mechanical*** process for computing new sentences

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

Inference 1: Forward Chaining

Forward Chaining

Based on rule of ***modus ponens***

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

Then can conclude Q

Backward Chaining: search

start from the query and go backwards

Analysis

- Sound?
- Complete?

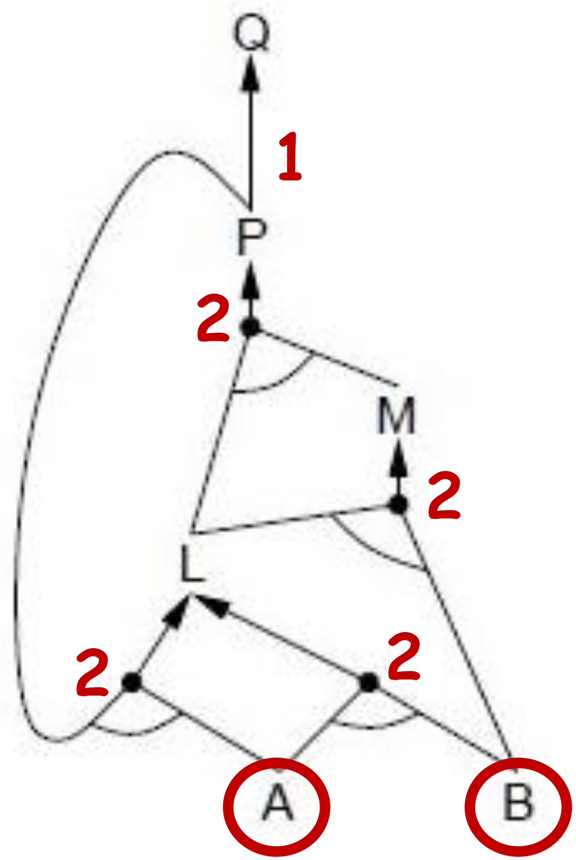
Can you prove

$$\{\,\} \models Q \vee \neg Q$$

- 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

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

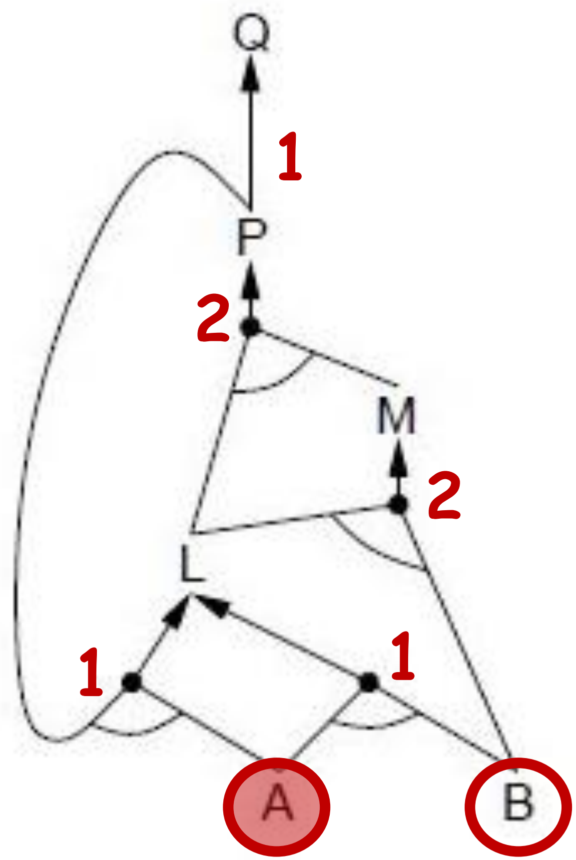
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

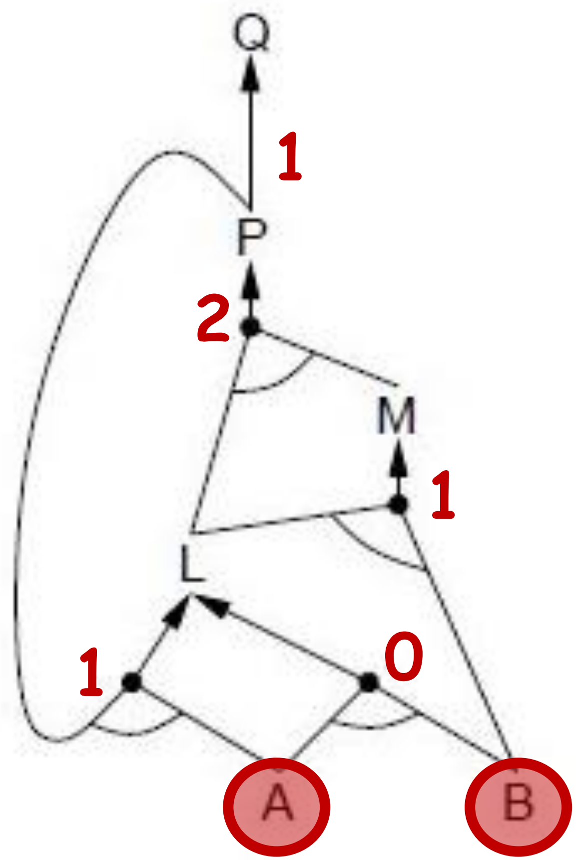
A

B



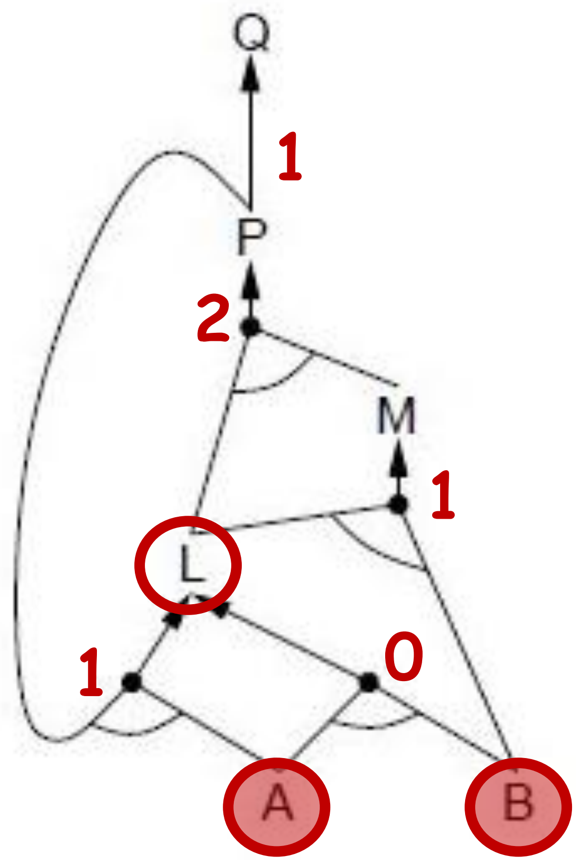
Example

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



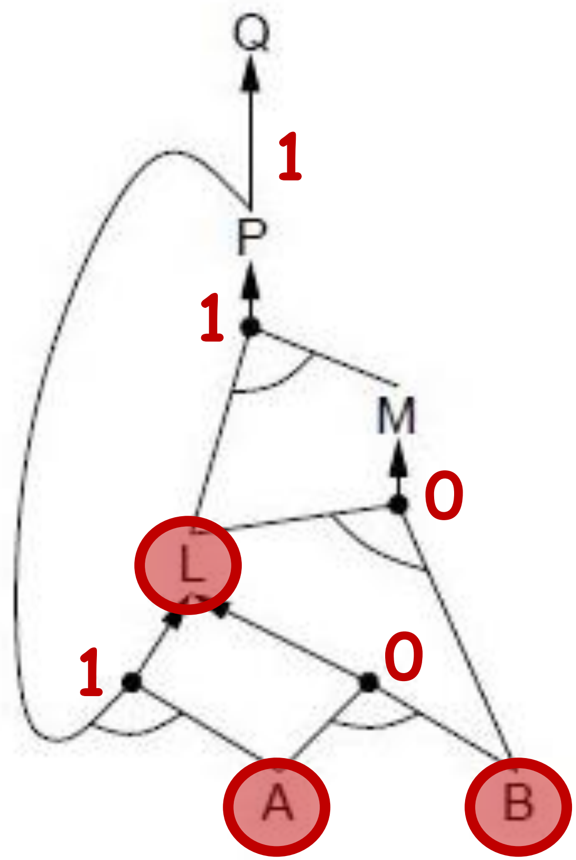
Example

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



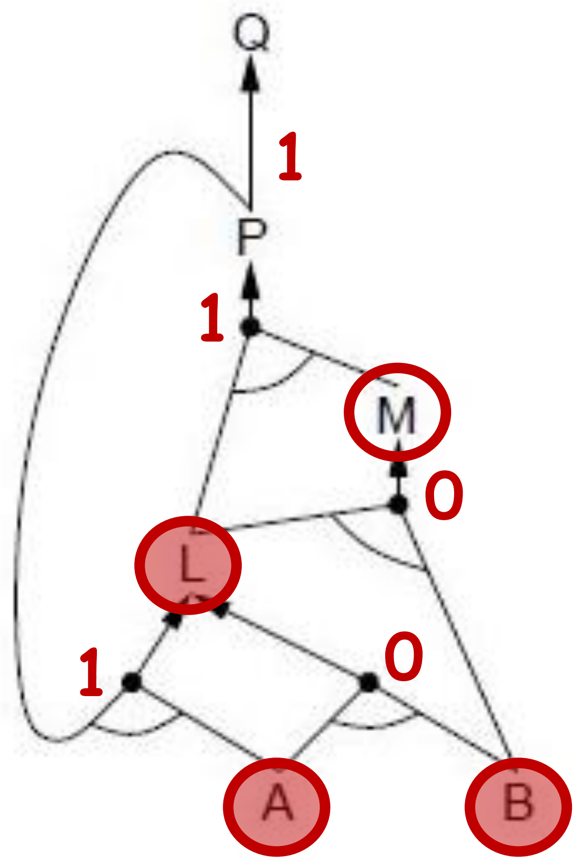
Example

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



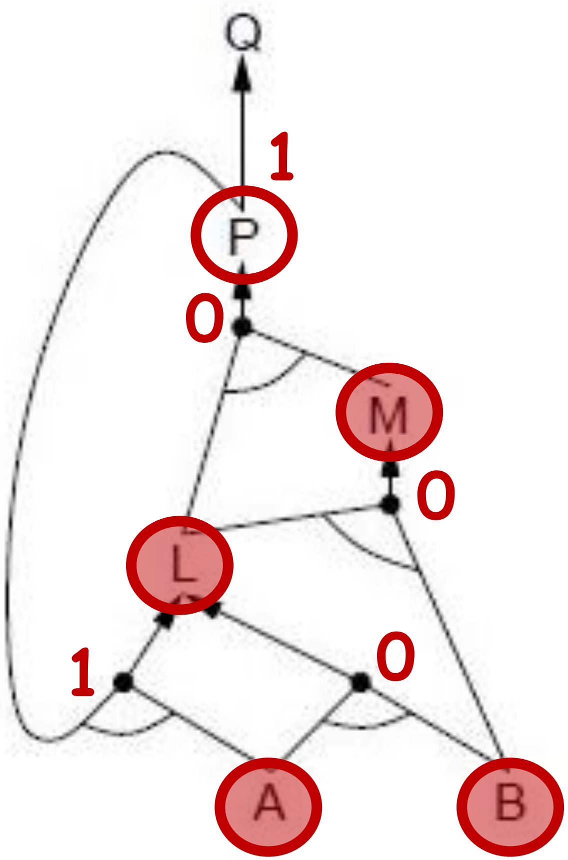
Example

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



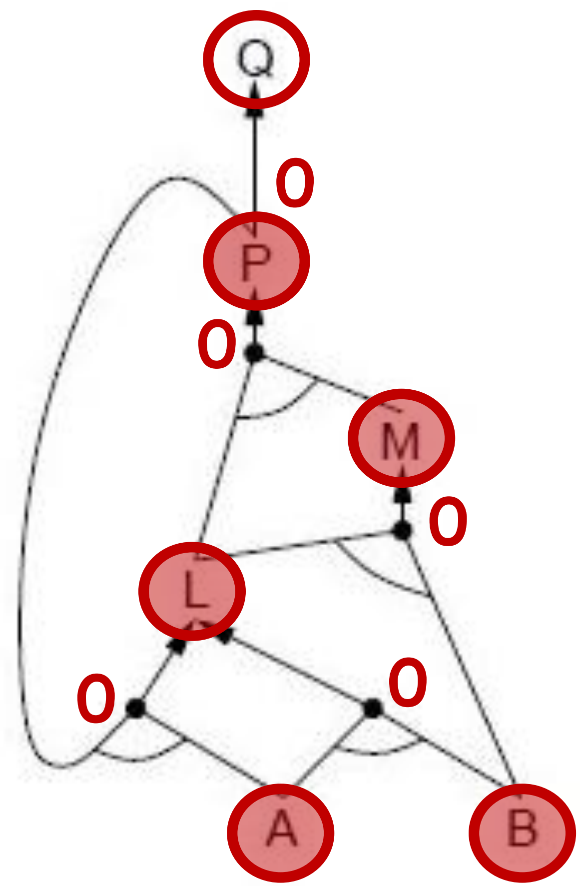
Example

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



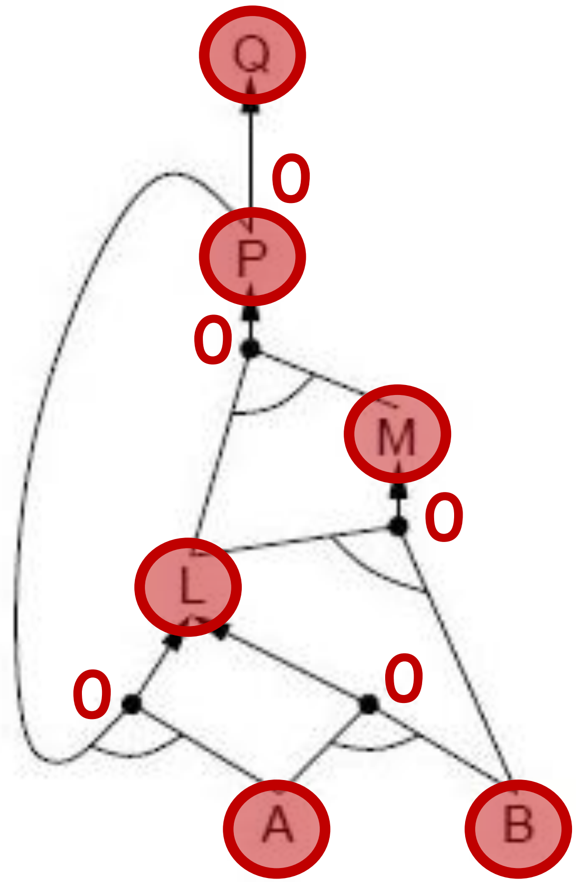
Example

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



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. GSAT
4. Davis Putnam

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]

$$\{ (p \vee \alpha), (\neg p \vee \beta \vee \gamma) \} \ \vdash_R (\alpha \vee \beta \vee \gamma)$$

Correctness

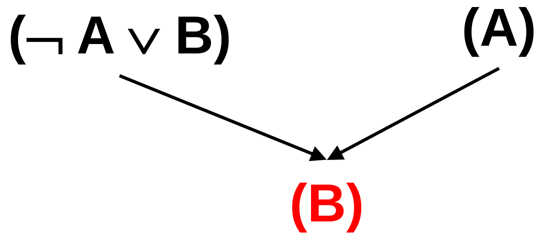
If S1 $\vdash_R$ S2 then S1 $\models$ S2

Refutation Completeness:

If S is unsatisfiable then S $\vdash_R$ ()

Resolution subsumes Modus Ponens

$A \rightarrow B, A \models B$



If Will goes, Jane will go

~W V J

If doesn't go, Jane will still go

W V J

Will Jane go?

|= J?

$\boxed{\text{J V J}}$ =J

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

~J ~W

~W V J □

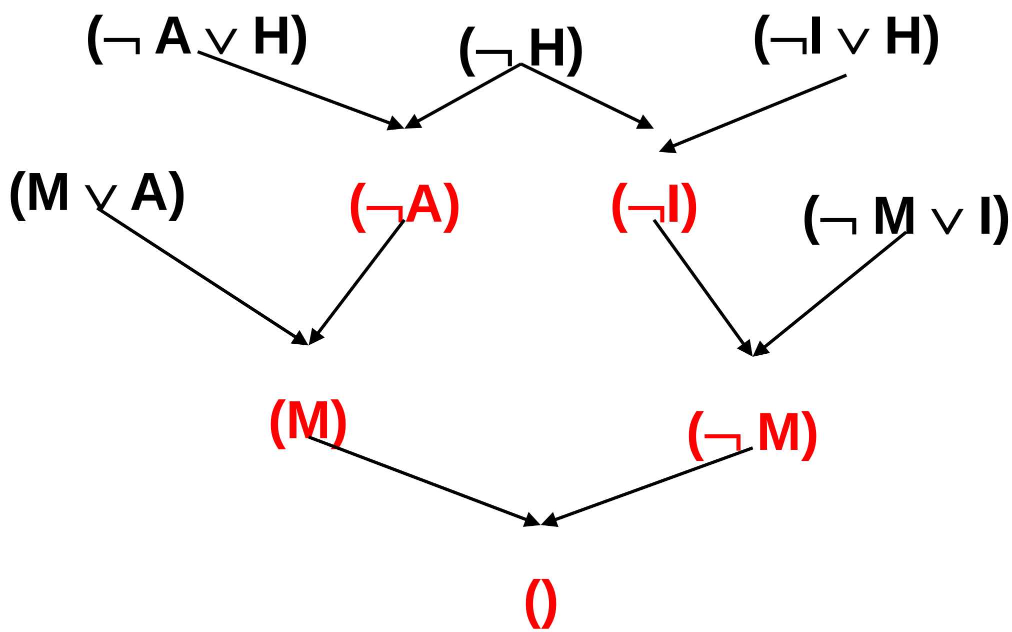
W V J J

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 => QVP)

Idea 1: Set of Support: *At least one of C1 or C2 must be either the goal clause or a clause derived by doing resolutions on the goal clause (*COMPLETE*)*

Idea 2: Linear input form: *Atleast one of C1 or C2 must be one of the clauses in the input KB (*INCOMPLETE*)*

Model Finding

- Find assignments to variables that makes a formula true

Inference 3: Model Enumeration

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

Inference 4: DPLL
(Enumeration of ***Partial*** Models)

[Davis, Putnam, Loveland & Logemann 1962]

Version 1

```
dpll_1(pa){
  if (pa makes F false) return false;
  if (pa makes F true) return true;
  choose P in F;
  if (dpll_1(pa ∪ {P=0})) return true;
  return dpll_1(pa ∪ {P=1});
}
```

Returns true if F is satisfiable, false otherwise

DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$

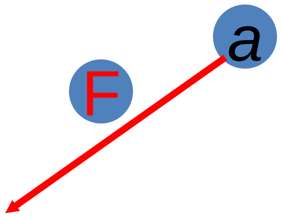
DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



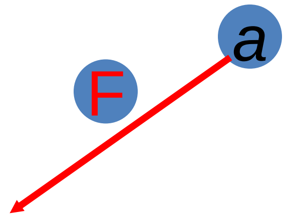
DPLL Version 1

$(\mathrm{F} \lor b \lor c)$

$(\mathrm{F} \lor \neg b)$

$(\mathrm{F} \lor \neg c)$

$(\mathrm{T} \lor c)$



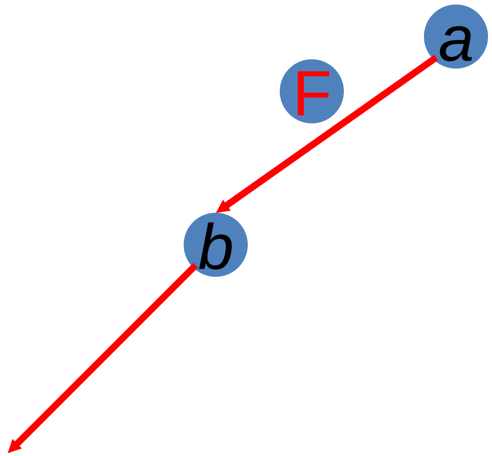
DPLL Version 1

$(F \lor F \lor c)$

$(F \lor T)$

$(F \lor \neg c)$

$(T \lor c)$



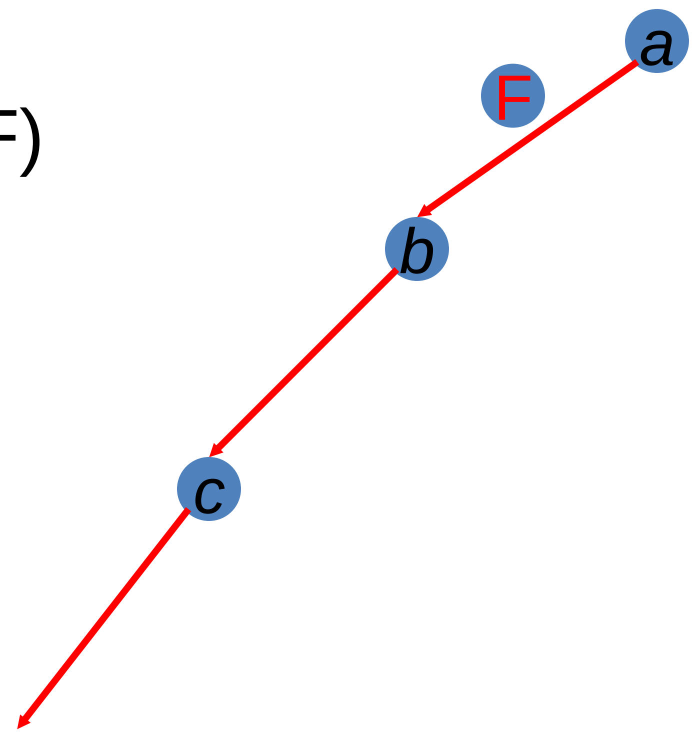
DPLL Version 1

(F ∨ F ∨ F)

(F ∨ T)

(F ∨ T)

(T ∨ F)



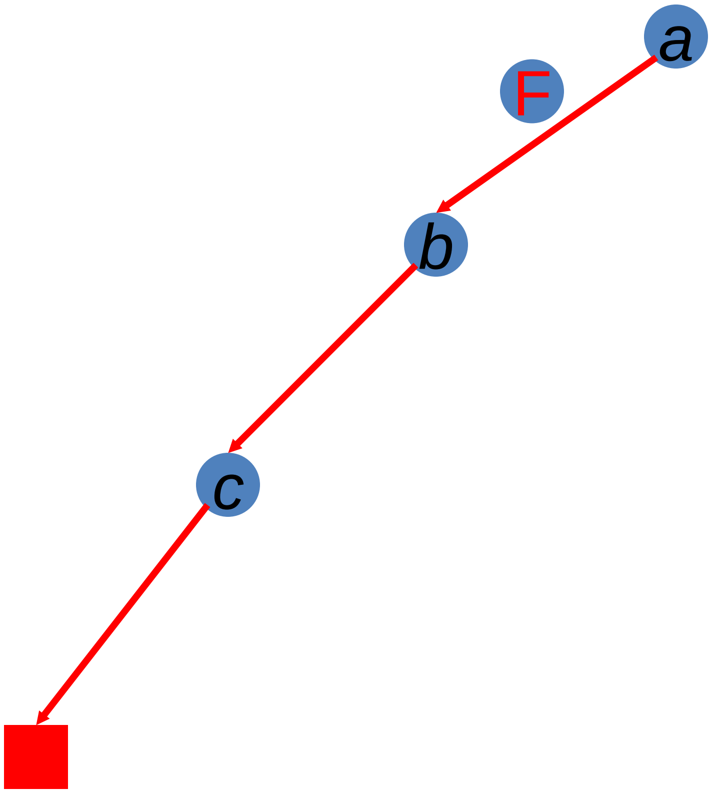
DPLL Version 1

F

T

T

T



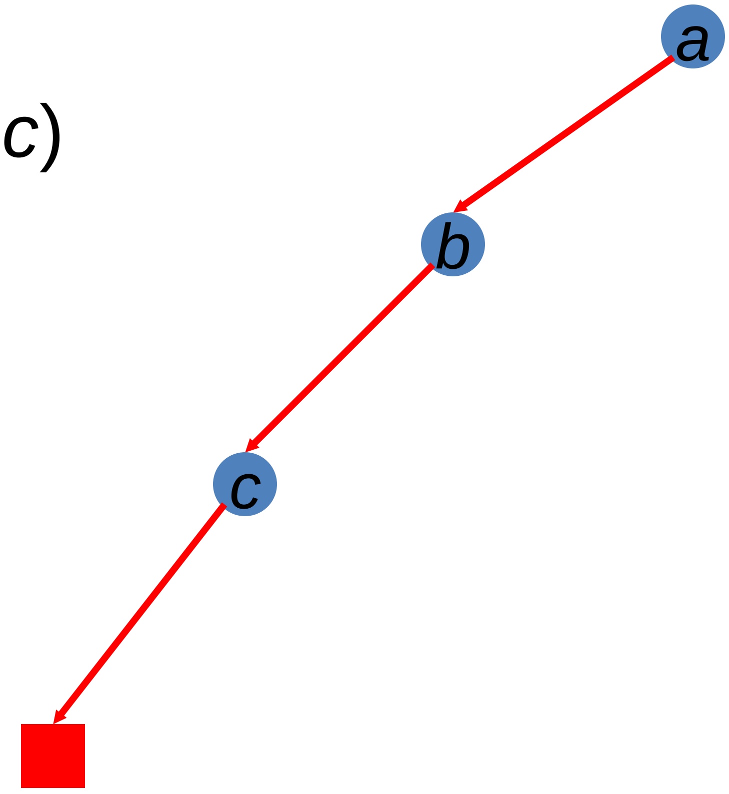
DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



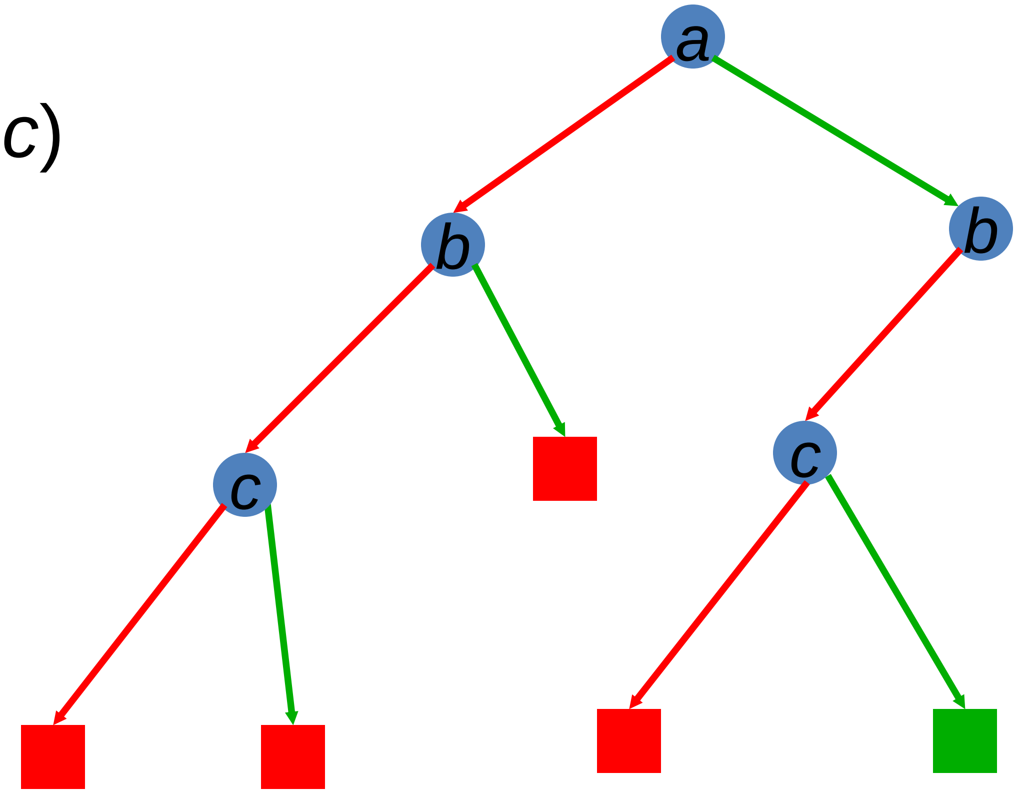
DPLL Version 1

$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$



DPLL as Search

- Search Space?
- Algorithm?

Improving DPLL

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

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

Therefore: Okay to delete clauses containing true literals!

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

Therefore: Okay to shorten clauses containing false literals!

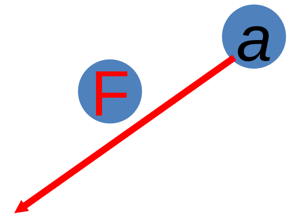
If literal L_1 is false, then clause (L_1) is false

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 ¬literal
  if (F contains empty clause)
     return false;
  choose V in F;
  if (dpll_2(F, ¬V))return true;
  return dpll_2(F, V);
}
```

DPLL Version 2



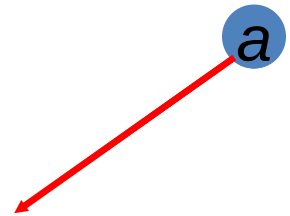
$(\mathrm{F} \vee b \vee c)$

$(\mathrm{F} \vee \neg b)$

$(\mathrm{F} \vee \neg c)$

$(\mathrm{T} \vee c)$

DPLL Version 2



$(b \vee c)$

$(\neg b)$

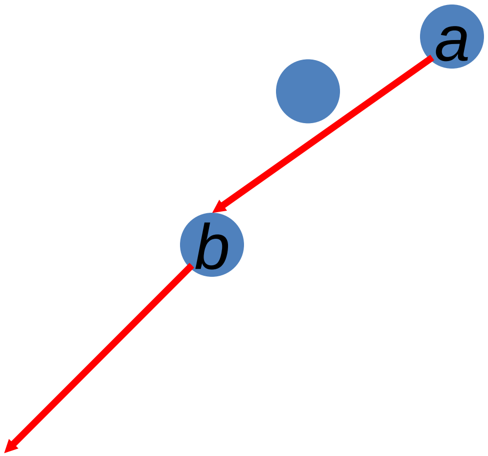
$(\neg c)$

DPLL Version 2

(F ∨ *c*)

(T)

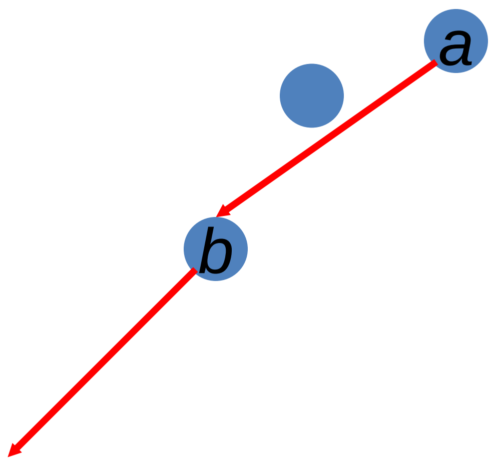
(¬*c*)



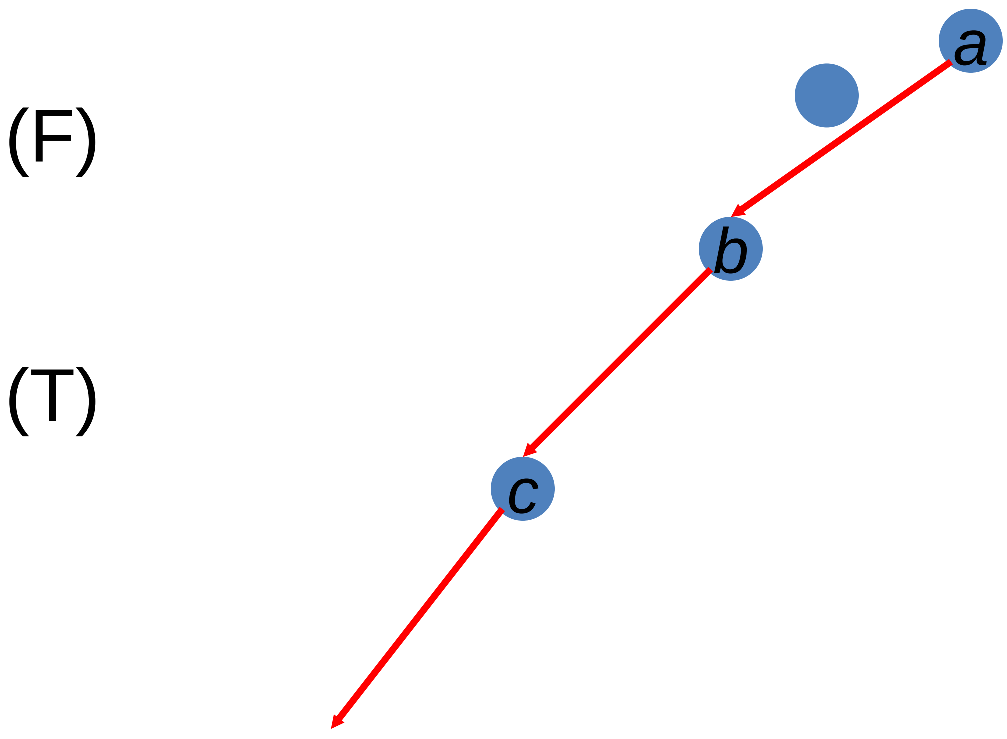
DPLL Version 2

(*c*)

(¬*c*)



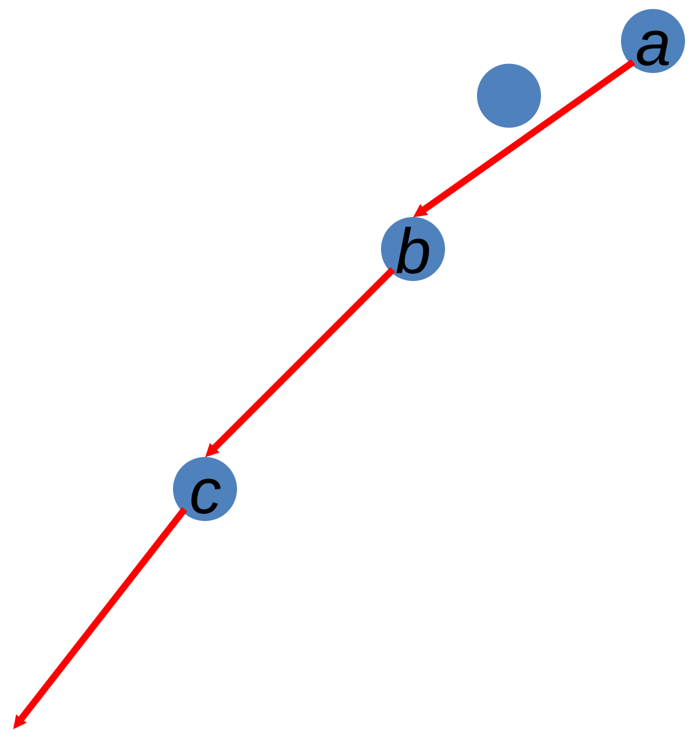
DPLL Version 2



DPLL Version 2

Empty clause!

()



Structure in Clauses

- Unit Literals (unit propagation)

  A literal that appears in a singleton clause

  {{¬b c}{¬c}{a ¬b e}{d b}{e a ¬c}}

  Might as well set it true! *And simplify*

  {{¬b} {a ¬b e}{d b}}

  {{d}}

- Pure Literals
  - A symbol that always appears with same sign
  - {{a ¬b c}{¬c d ¬e}{¬a ¬b e}{d b}{e a ¬c}}

  Might as well set it true! *And simplify*

  {{a ¬b c} {¬a ¬b e} {e a ¬c}}

In Other Words

Formula $(L) \wedge C_2 \wedge C_3 \wedge \ldots$ 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 $(L) \wedge C_2 \wedge C_3 \wedge \ldots$ 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

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬literal
  if (F contains empty clause)
     return false;


  choose V in F;
  if (dpll(F, ¬V))return true;
  return dpll(F, V);
}
```

DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
  remove clauses containing literal
  if (F contains no clauses) return true;
  shorten clauses containing ¬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, ¬V))return true;
  return dpll(F, V);
}
```

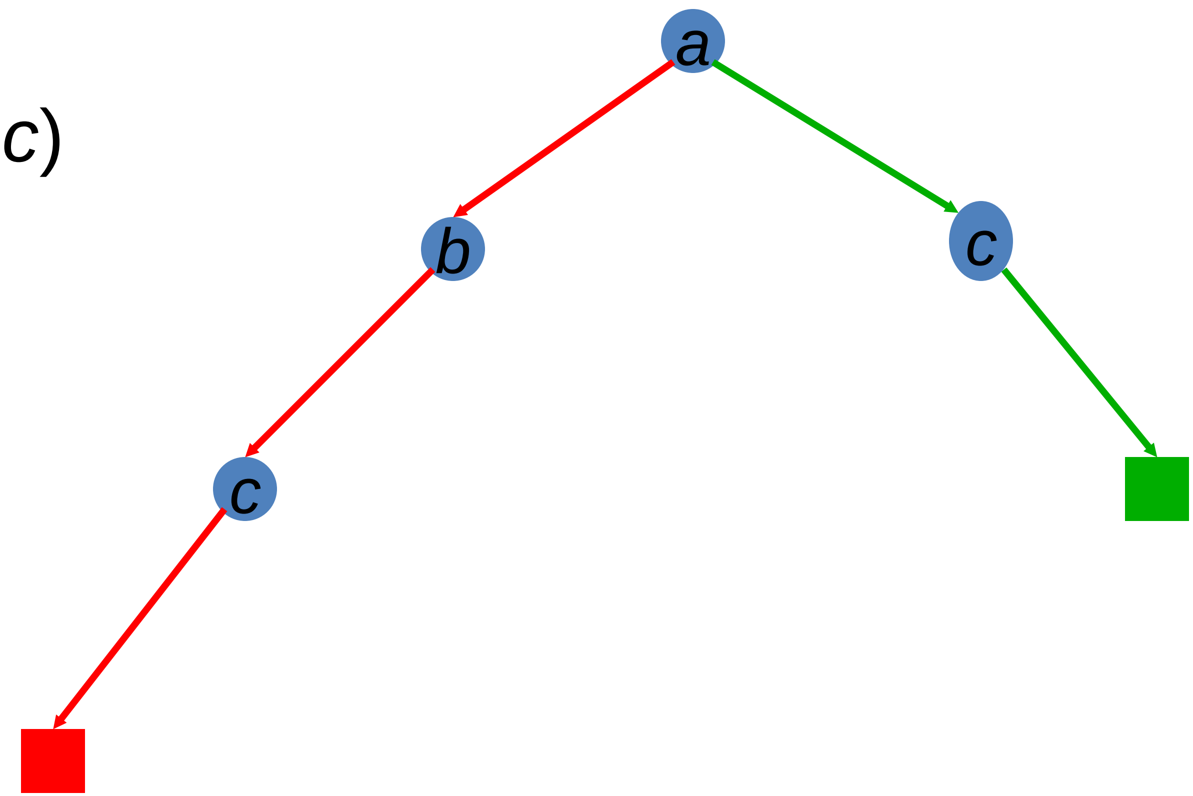
DPLL (for real)

$(a \lor b \lor c)$

$(a \lor \neg b)$

$(a \lor \neg c)$

$(\neg a \lor c)$



DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
   remove clauses containing literal
   if (F contains no clauses) return true;
   shorten clauses containing ¬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, ¬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

Success of DPLL

- 1962 – DPLL invented
- 1992 – 300 propositions
- 1997 – 600 propositions (satz)
- Additional techniques:
  - Learning conflict clauses at backtrack points
  - Randomized restarts
  - 2002 (zChaff) 1,000,000 propositions – encodings of hardware verification problems

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→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 := argmin_x { break[x] | x is in clause };
    endif
    state[var] := 1 – state[var];
end
return state;
```

Put everything inside of a restart loop.
Parameters: p, max_flips, max_runs

Advs of WalkSAT over GSAT

- WalkSat guaranteed to make at least 1 false clause (in random walk also)

- Number of evaluations small per move
  - does not iterate over all variables
  - only variables in the sampled clause