CSE 473 Propositional Logic SAT Algorithms

Luke Zettlemoyer

(With many slides from Dan Weld, Raj Rao, Mausam, Stuart Russell, Dieter Fox, Henry Kautz, Min-Yen Kan…)

Irrationally held truths may be more harmful than reasoned errors.

- Thomas Huxley (1825-1895)

Propositional Logic

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

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Types of Reasoning (Inference)

- Deduction (showing **entailment**, |=)

  S = question

  Prove that KB |= S

  Typically use rules to derive new formulas from old (inference)

- Model Finding (showing **satisfiability**)

  S = description of problem

  Show S is satisfiable

Validity and Satisfiability

A sentence is valid if it is true in **all** models,

e.g., $True$, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in **some** model

e.g., $A \lor B$, C

A sentence is unsatisfiable if it is true in **no** models

e.g., $A \land \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \land \neg\alpha)$ is unsatisfiable

i.e., prove α by *reductio ad absurdum*

Inference

$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.
Entailment = needle in haystack; inference = finding it

Soundness: i is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB.

Truth Tables for Inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

Problem: exponential time and space!

Logical Equivalence

Two sentences are logically equivalent iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{array}{rcll}
(\alpha \land \beta) & \equiv & (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv & (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv & (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv & (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \lor \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv & (\neg\alpha \lor \neg\beta) & \text{De Morgan} \\
\neg(\alpha \lor \beta) & \equiv & (\neg\alpha \land \neg\beta) & \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv & ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv & ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{distributivity of } \lor \text{ over } \land
\end{array}
$$

Proof Methods

Proof methods divide into (roughly) two kinds:

Application of inference rules

– Legitimate (sound) generation of new sentences from old
– Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.
– Typically require translation of sentences into a normal form

Model checking

truth table enumeration (always exponential in n)
improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
heuristic search in model space (sound but incomplete)
  e.g., min-conflicts-like hill-climbing algorithms

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 Algorithms

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)

} **Deduction**

3. Exhaustive Enumeration
4. DPLL (Davis, Putnam Loveland & Logemann)
5. GSAT

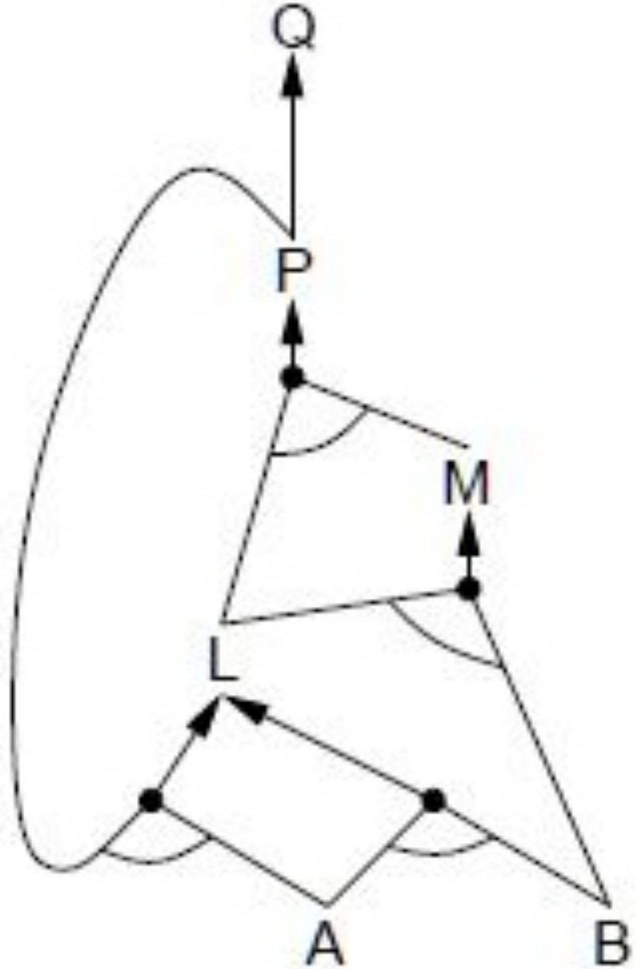
} **Model Finding**

Example

KB with Horn Clauses

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

Proof And/Or Graph



Inference Technique II: Forward/ Backward Chaining

- Require sentences to be in **Horn Form**:

  KB = conjunction of Horn clauses

  – Horn clause =
    - proposition symbol or
    - "(conjunction of symbols) $\Rightarrow$ symbol"

      (i.e. clause with at most 1 positive literal)

  – E.g., KB = $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

- F/B chaining based on "Modus Ponens" rule:

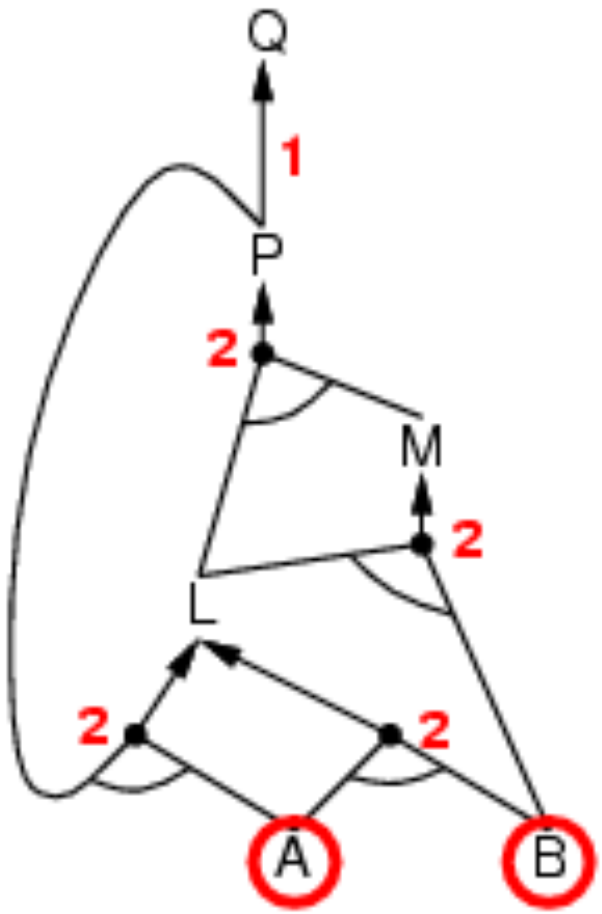
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{B}$$

  – Sound and complete for Horn clauses

Forward chaining algorithm

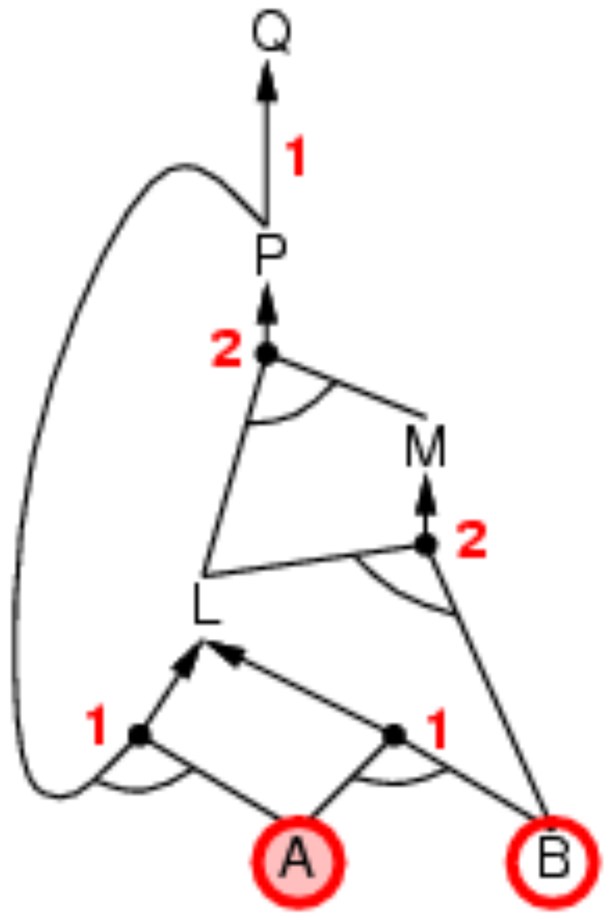
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true

  while agenda is not empty do
      p ← POP(agenda)
      unless inferred[p] do
          inferred[p] ← true
          for each Horn clause c in whose premise p appears do
              decrement count[c]
              if count[c] = 0 then do
                  if HEAD[c] = q then return true
                  PUSH(HEAD[c], agenda)
  return false
```

Forward chaining example

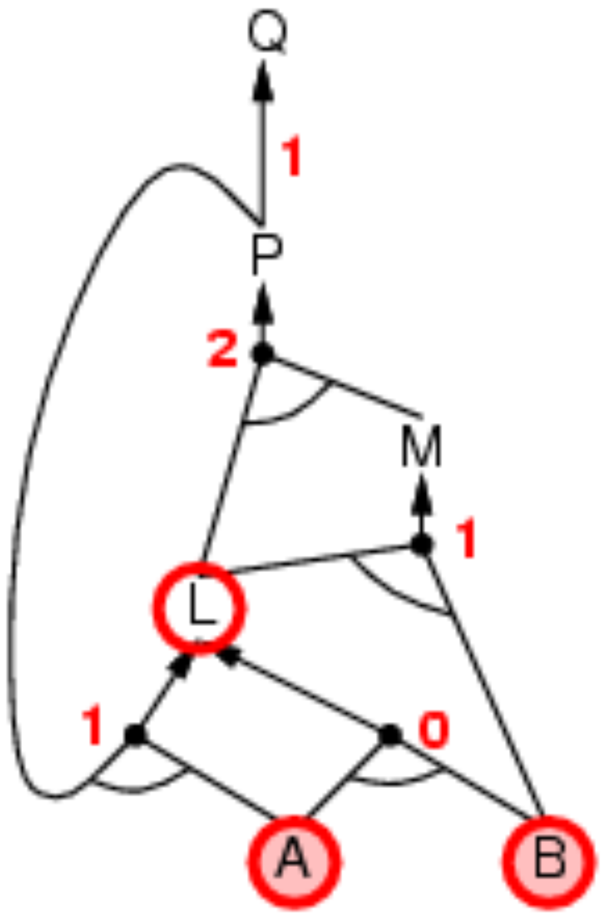


Query = Q
(i.e. "Is Q true?")

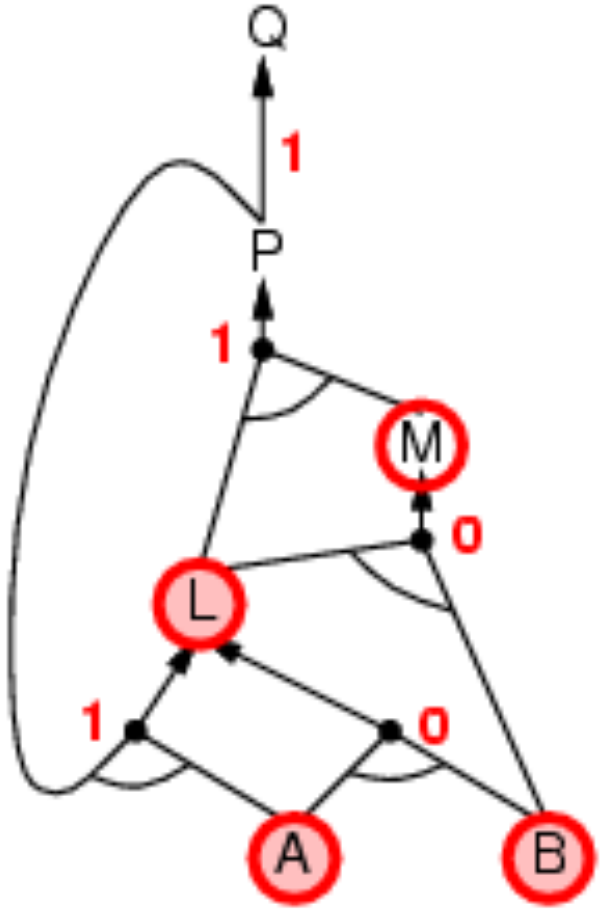
Forward chaining example



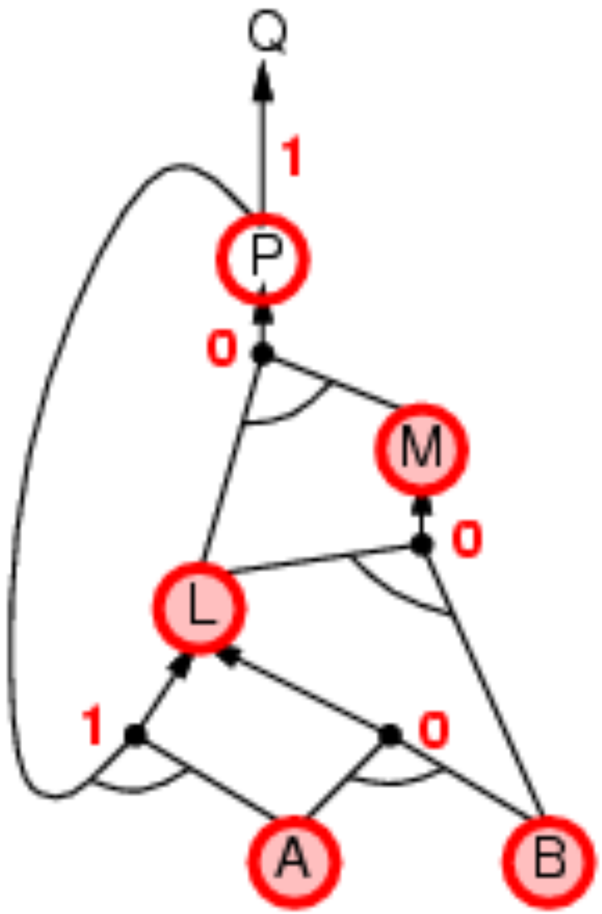
Forward chaining example



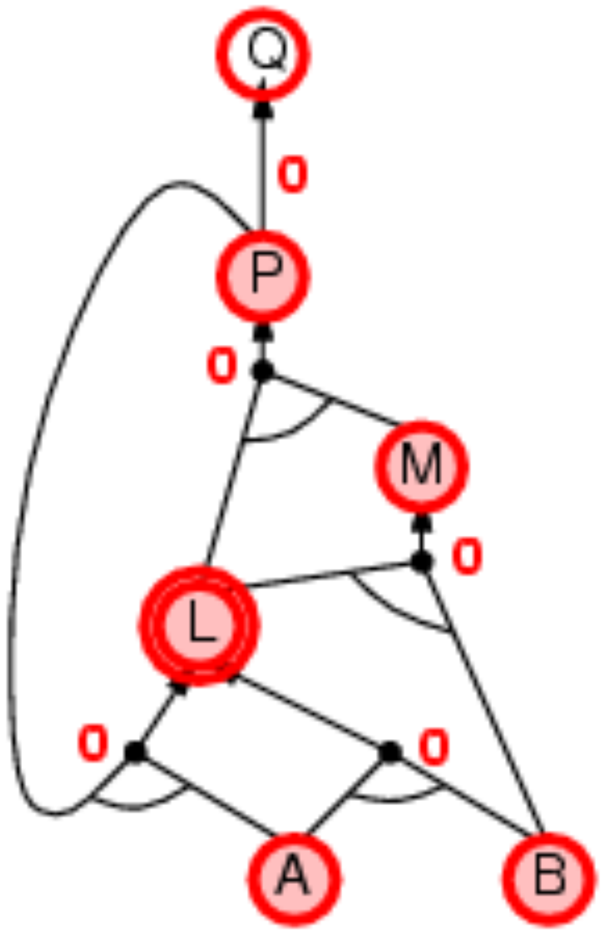
Forward chaining example



Forward chaining example



Forward chaining example



Backward chaining

Idea: work backwards from the query *q*:

to prove *q* by BC,

check if *q* is known already, or

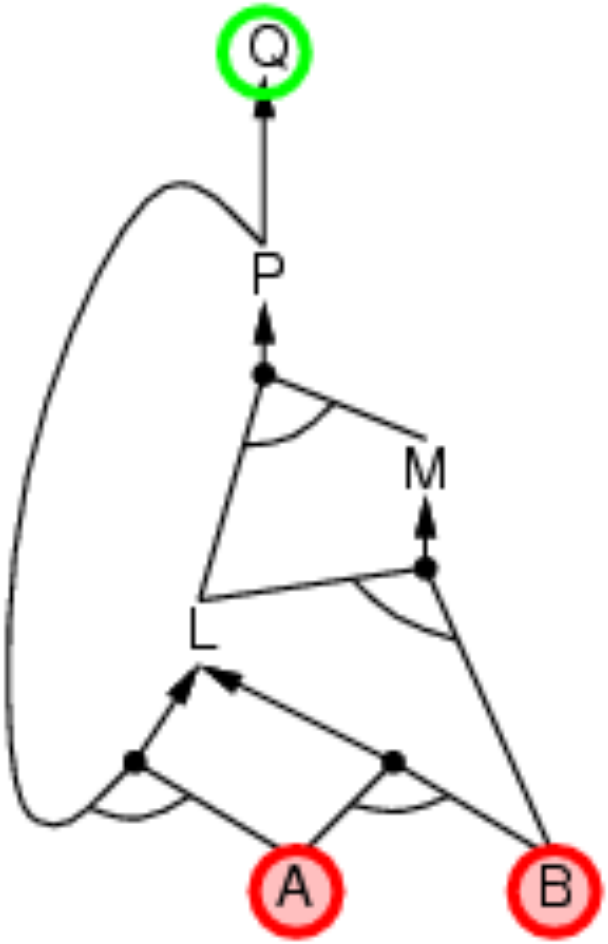
prove by BC all premises of some rule concluding *q*

Avoid loops: check if new subgoal is already on goal stack

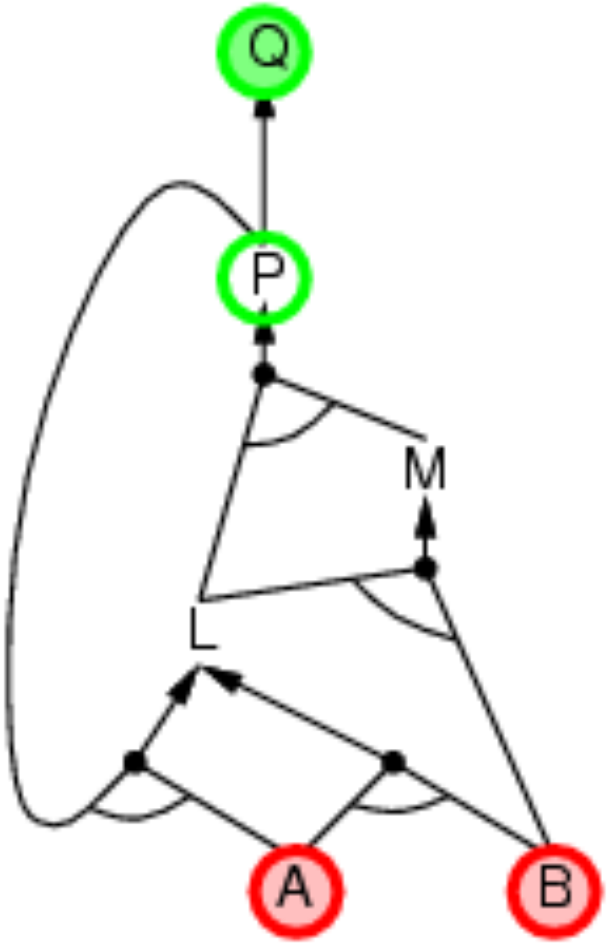
Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

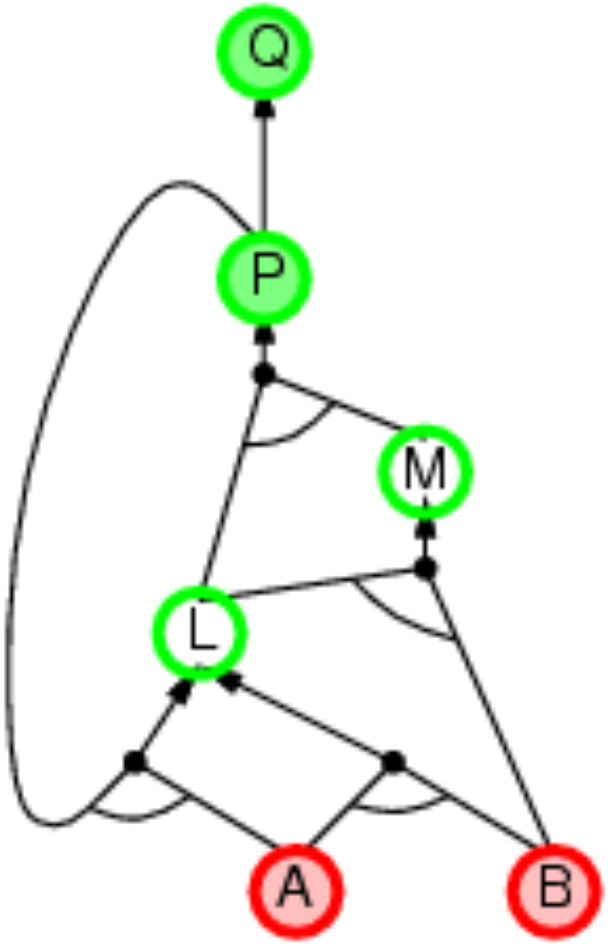
Backward chaining example



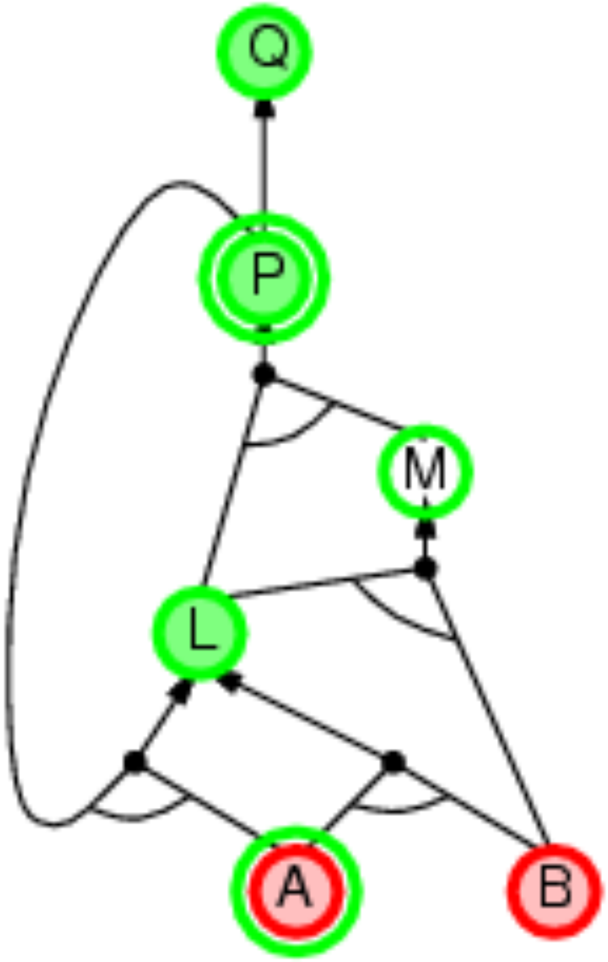
Backward chaining example



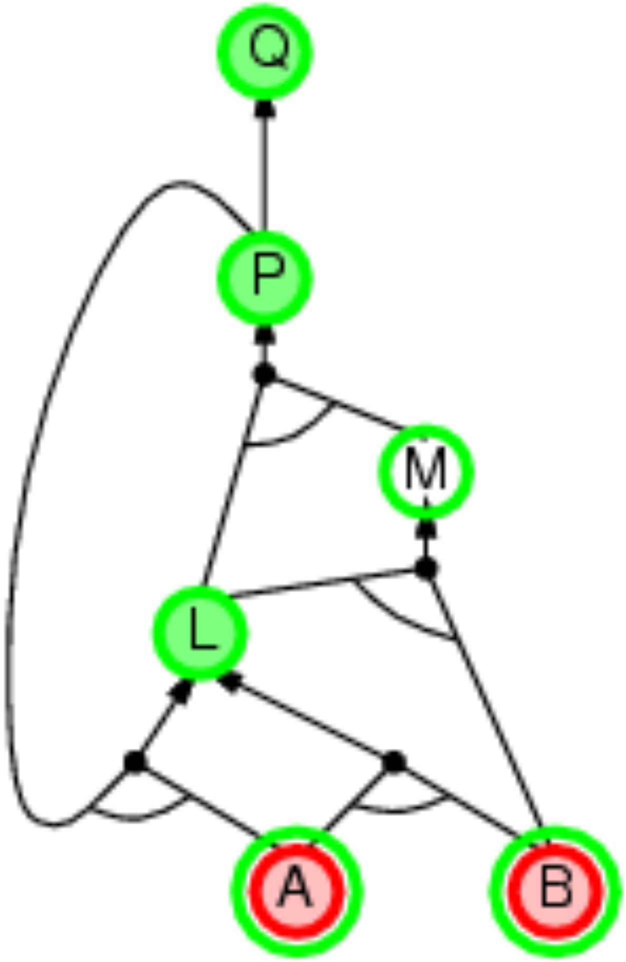
Backward chaining example



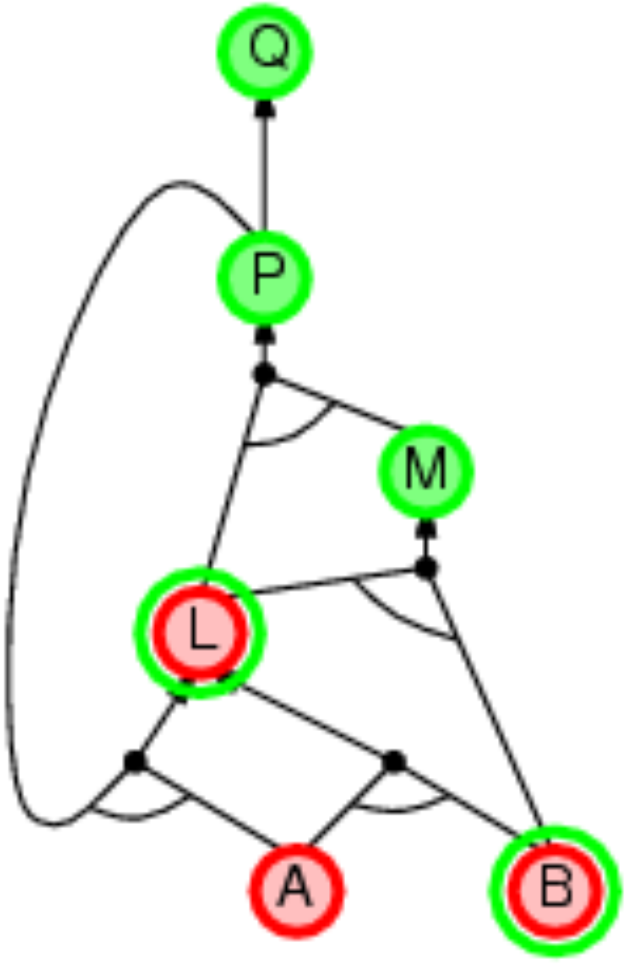
Backward chaining example



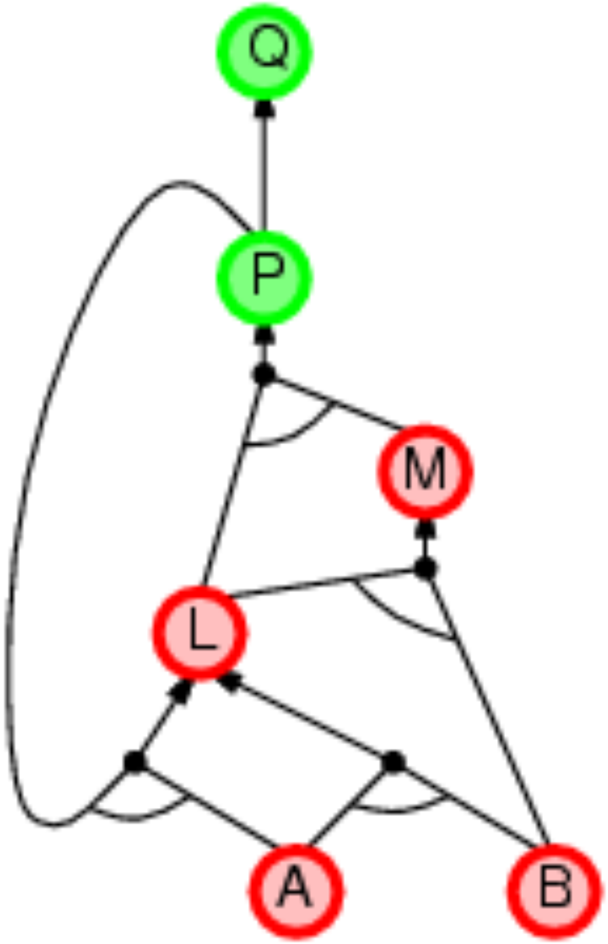
Backward chaining example



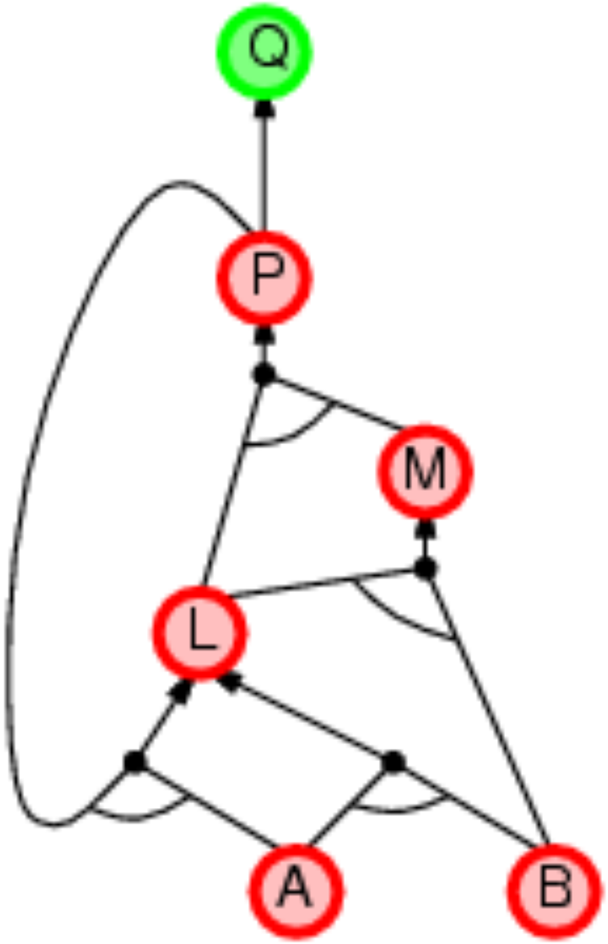
Backward chaining example



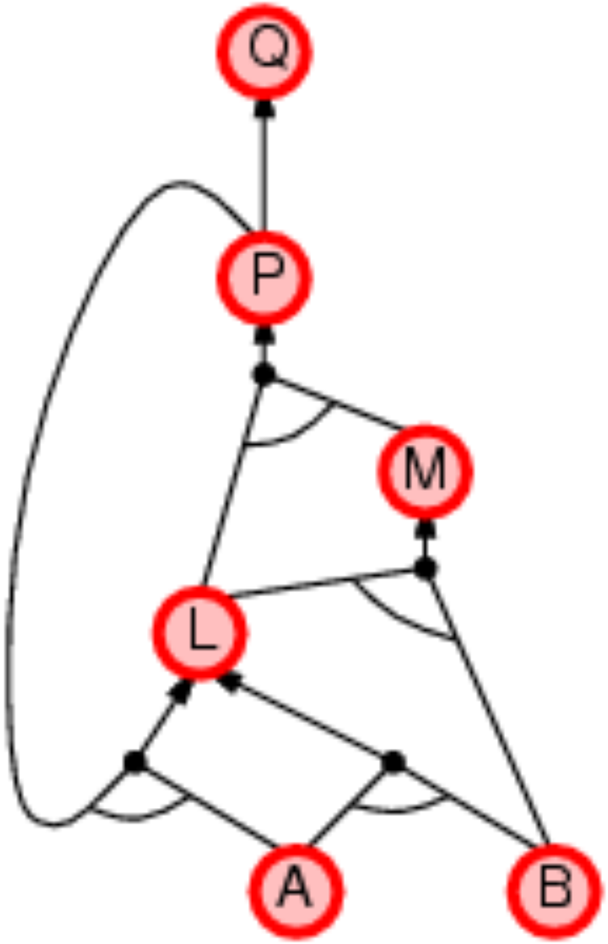
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing,
  - e.g., object recognition, routine decisions
- FC may do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving,
  - e.g., How do I get an A in this class?
  - e.g., What is my best exit strategy out of the classroom?
  - e.g., How can I impress my date tonight?
- Complexity of BC can be much less than linear in size of KB

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 $\vdash_R$ ()

Conversion to CNF

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

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:

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

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

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

Resolution algorithm

- To show **KB** $\vdash \alpha$, use proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

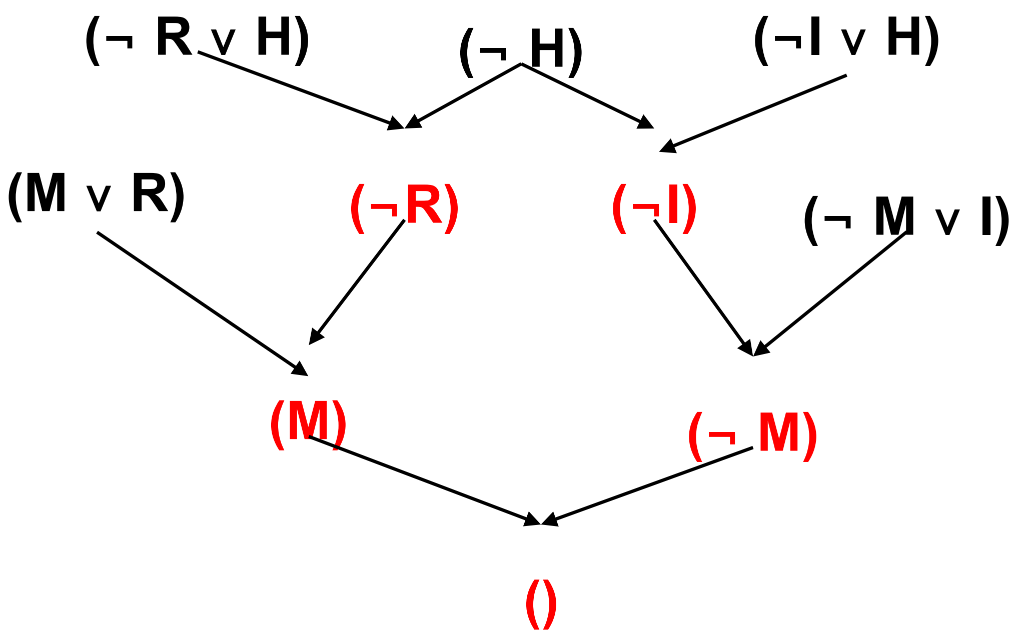
```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each C_i, C_j in clauses do
            resolvents ← PL-RESOLVE(C_i, C_j)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Resolution

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

Prove: the unicorn is horned.

M = mythical
I = immortal
R = reptile
H = horned



Resolution as Search

- States?
- Operators

Model Checking: Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*false*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*true*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

alpha_1 = not P_{12} ("[1,2] is safe")

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

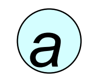
$(a \vee b \vee c)$

$(a \vee \neg b)$

$(a \vee \neg c)$

$(\neg a \vee c)$

DPLL Version 1



$(a \lor b \lor c)$

$(a \lor \neg b)$

$(a \lor \neg c)$

$(\neg a \lor c)$

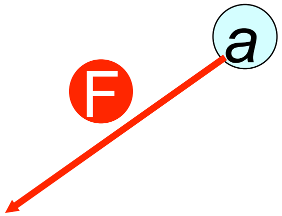
DPLL Version 1

(F ∨ *b* ∨ *c*)

(F ∨ ¬*b*)

(F ∨ ¬*c*)

(T ∨ *c*)



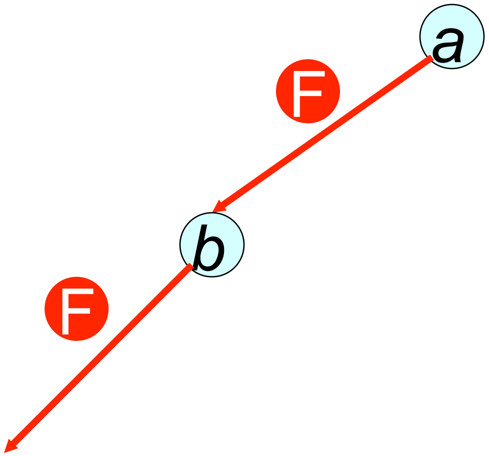
DPLL Version 1

(F ∨ F ∨ *c*)

(F ∨ T)

(F ∨ ¬*c*)

(T ∨ *c*)



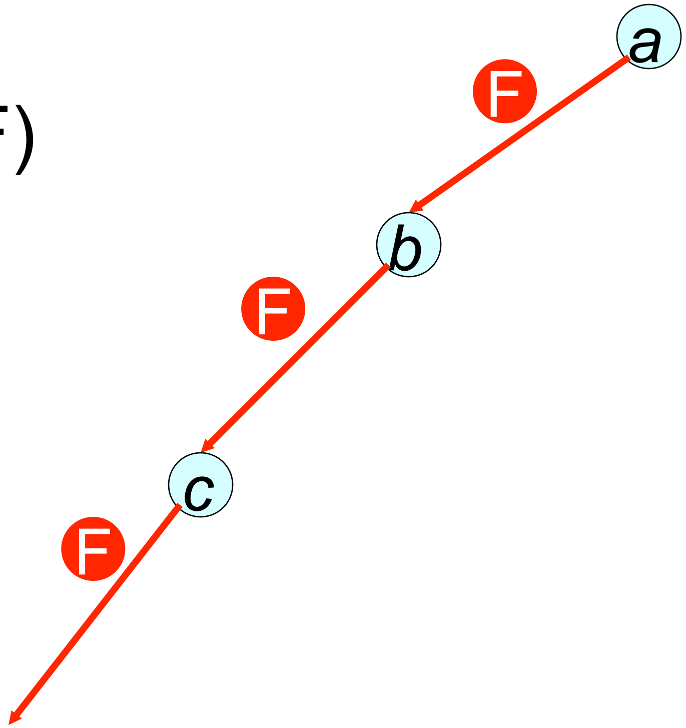
DPLL Version 1

(F v F v F)

(F v T)

(F v T)

(T v F)



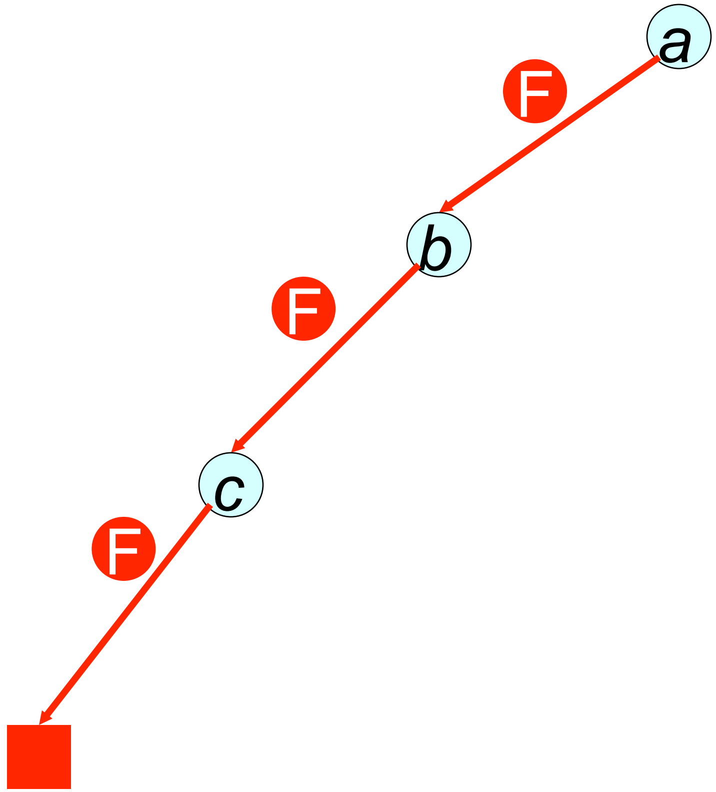
DPLL Version 1

F

T

T

T



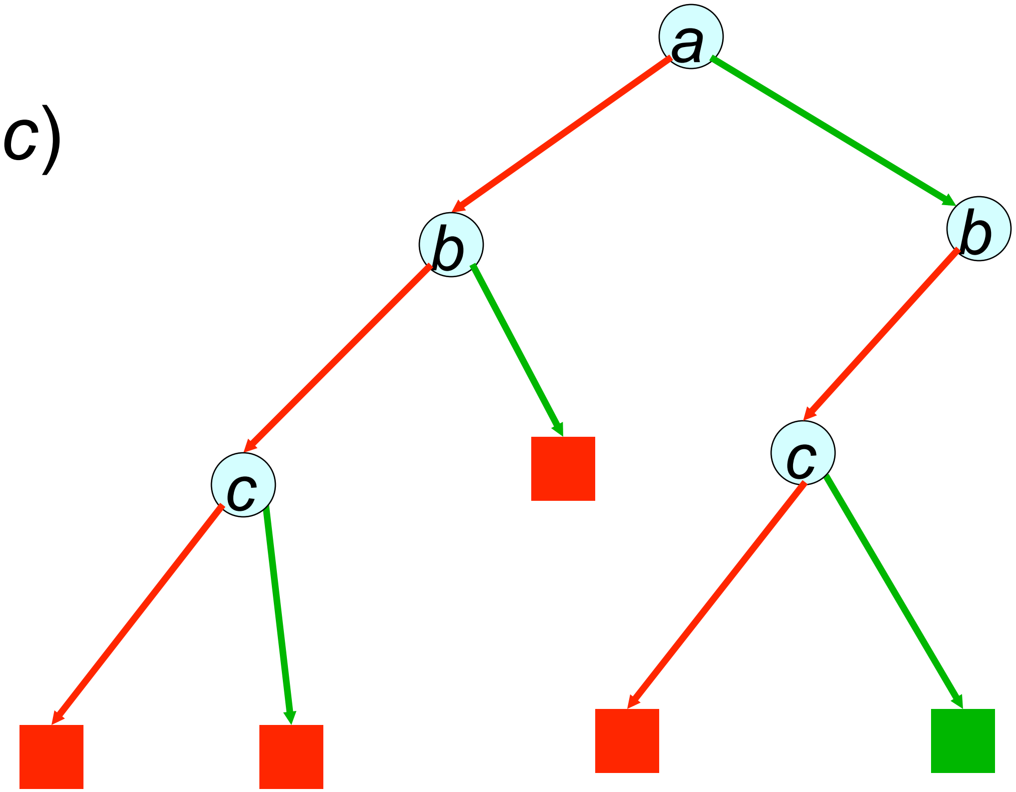
DPLL Version 1

$(a \lor b \lor c)$

$(a \lor \neg b)$

$(a \lor \neg c)$

$(\neg a \lor c)$



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!

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 delete shorten containing false literals!

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 delete shorten containing false literals!

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

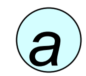
Therefore: the empty clause means false!

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);
}
```

Partial assignment corresponding to a node is the set of chosen literals on the path from the root to the node

DPLL Version 2



$(a \lor b \lor c)$

$(a \lor \neg b)$

$(a \lor \neg c)$

$(\neg a \lor c)$

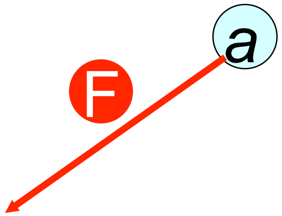
DPLL Version 2

(F ∨ *b* ∨ *c*)

(F ∨ ¬*b*)

(F ∨ ¬*c*)

(T ∨ *c*)

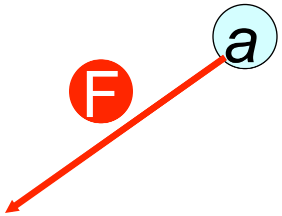


DPLL Version 2

$(b \vee c)$

$(\neg b)$

$(\neg c)$

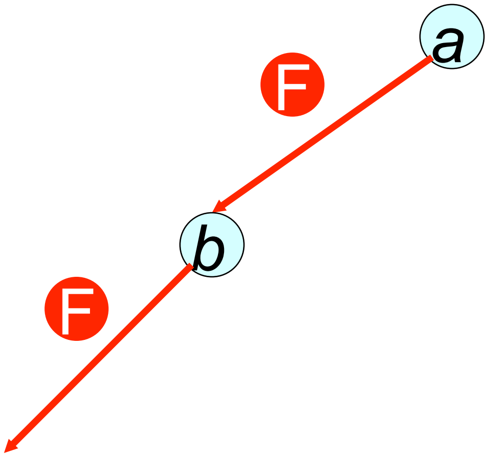


DPLL Version 2

(F ∨ *c*)

(T)

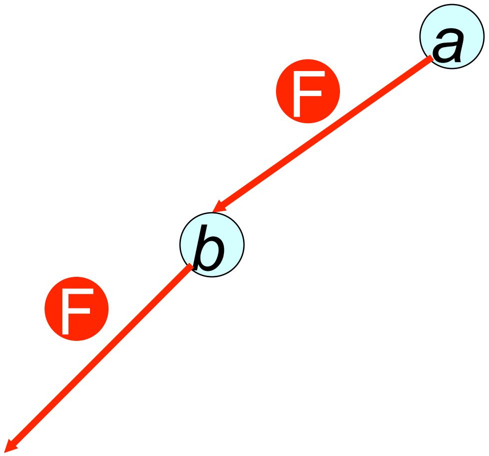
(¬*c*)



DPLL Version 2

(c)

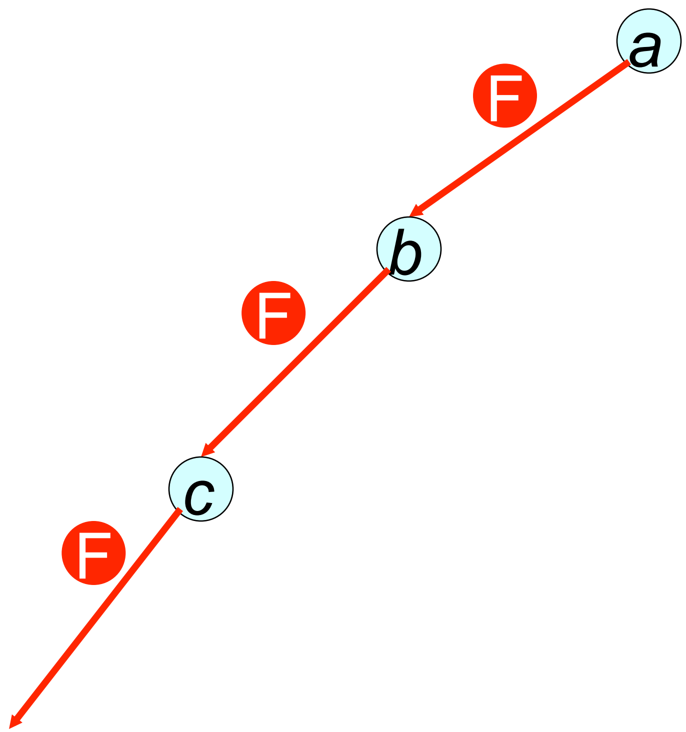
$(\neg c)$



DPLL Version 2

(F)

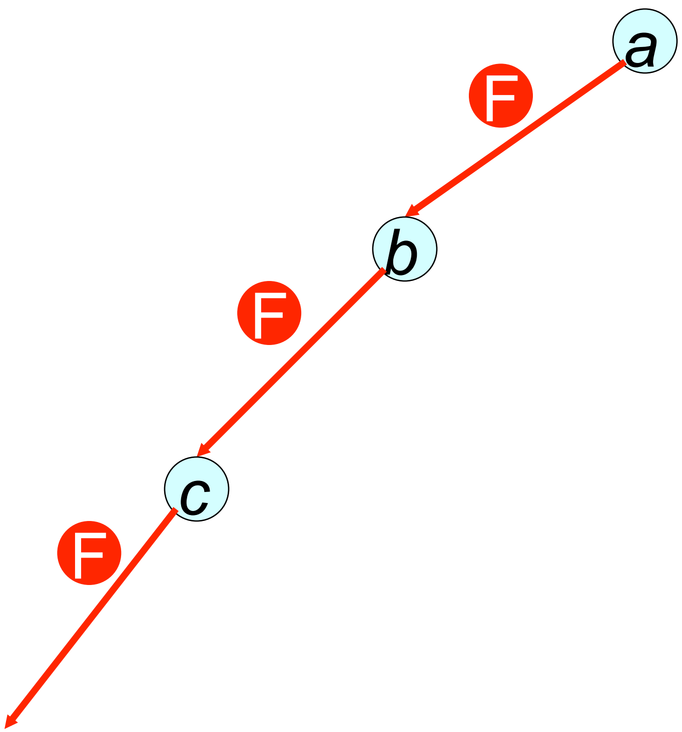
(T)



DPLL Version 2

Empty clause!

()



Representing Formulae

- CNF = Conjunctive Normal Form
  - Conjunction (∧) of Disjunctions (∨)
- Represent as set of sets
  - ((A, B), (¬A, C), (¬C))
  - ((¬A), (A))
  - (())
  - ((A))
  - ()

Structure in Clauses

- Unit Literals

  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}}

- 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 dpll(F, L);
  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);
}
```

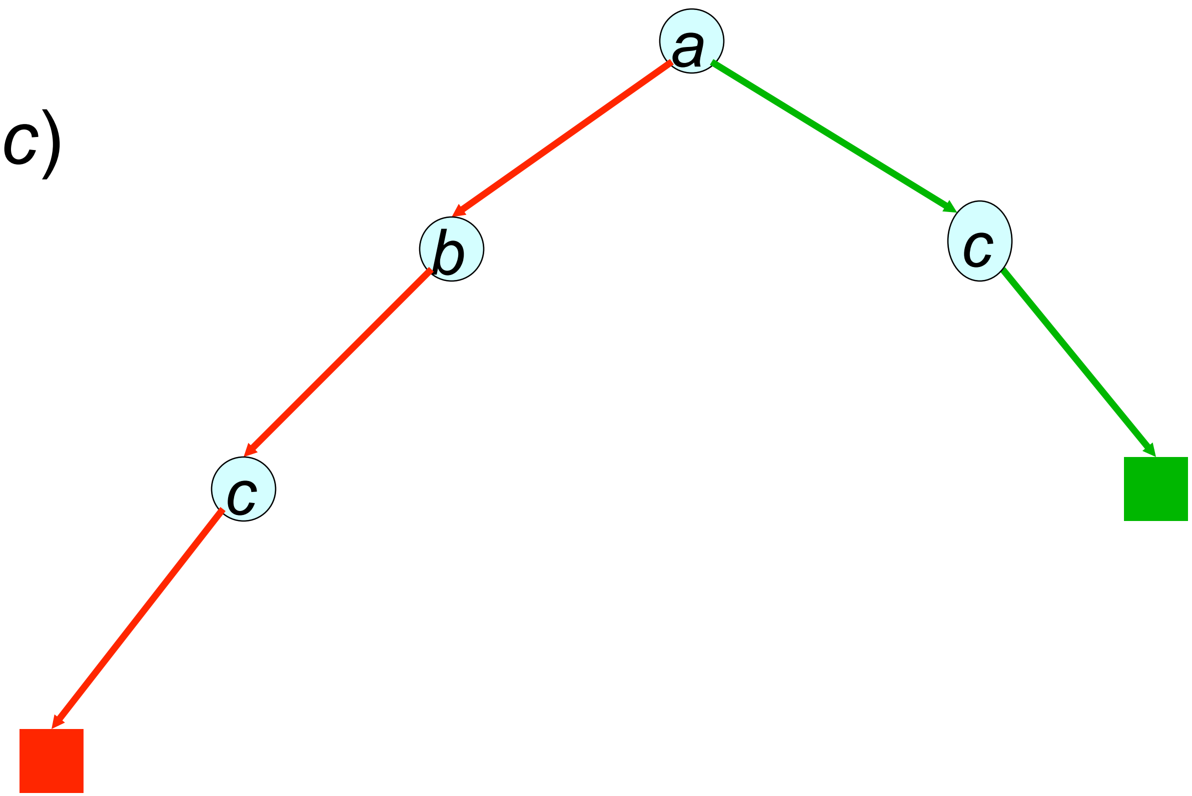
DPLL (for real)

$(a \lor b \lor c)$

$(a \lor \neg b)$

$(a \lor \neg c)$

$(\neg a \lor c)$



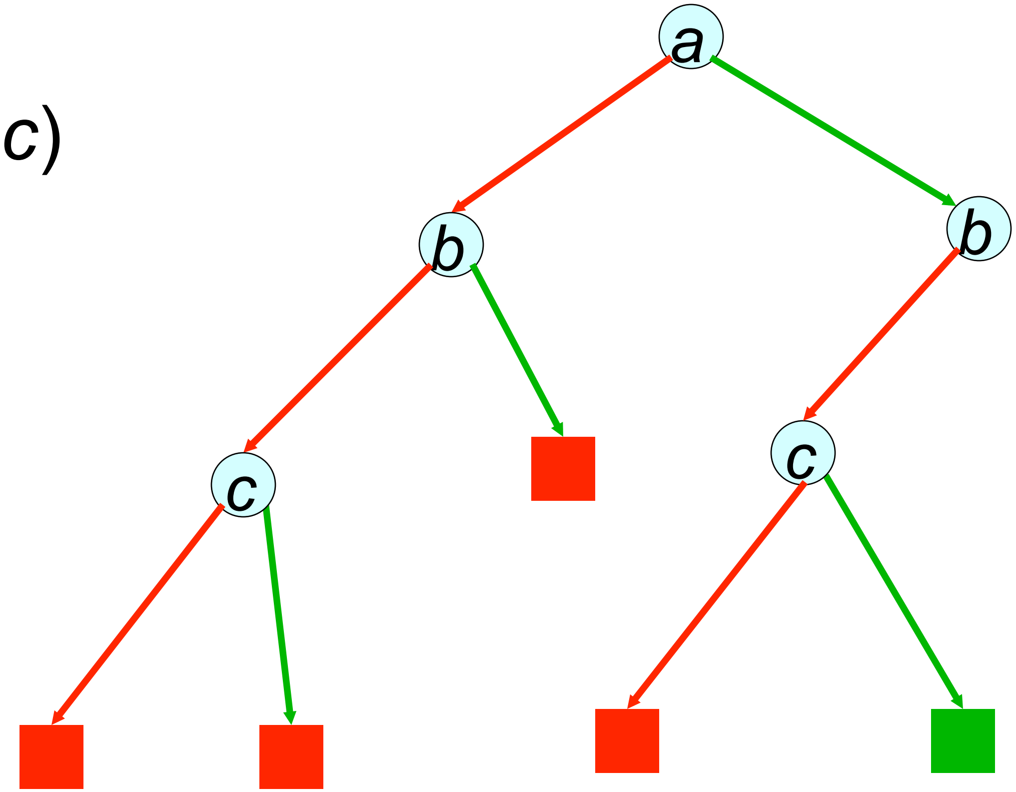
Compare with DPLL Version 1

($a \vee b \vee c$)

($a \vee \neg b$)

($a \vee \neg c$)

($\neg a \vee c$)



DPLL (for real!)

Davis – Putnam – Loveland – Logemann

```
dpll(F, literal){
   remove clauses containing literal
   if (F contains no clauses) return true;
   shorten clauses containing ¬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:
  - **M**ost **o**ccurrences in clauses of **m**inimum 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

Other Ideas?

- How else could we solve SAT problems?

WalkSat (Take 1)

- ***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

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:

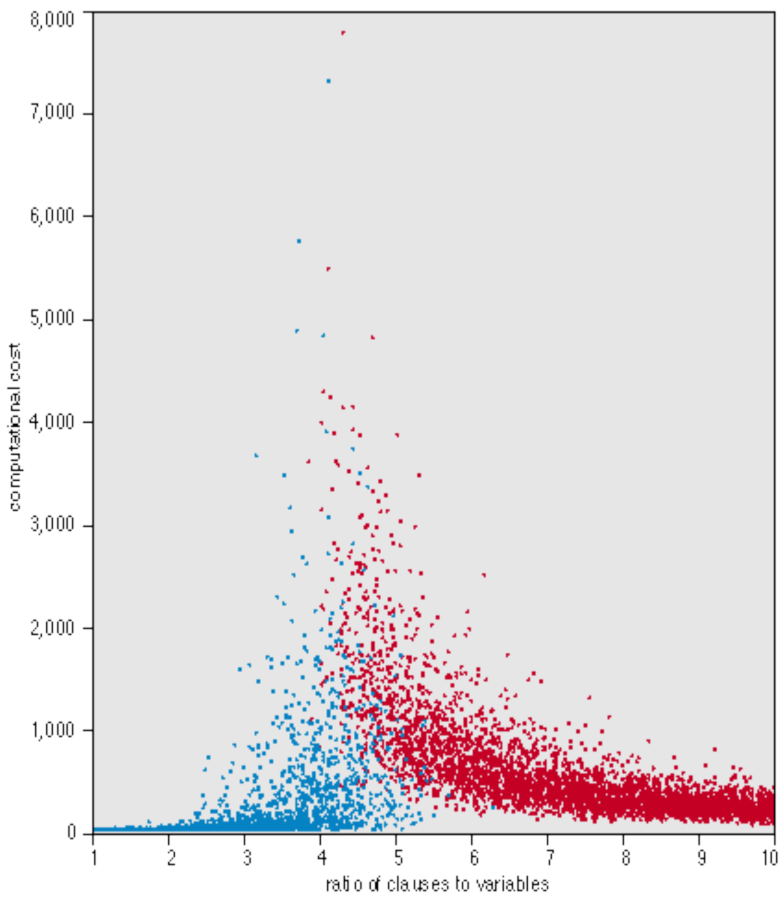
  #unsat(s2) = #unsat(s1) – make(s1,x) + break(s1,x)
- Idea 1: if a choice breaks nothing, it's likely good!
- Idea 2: near the solution, only the break count matters
  - the make count is usually 1

Walksat (Take 2)

```
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
            var := arg x min { 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

Random 3-SAT

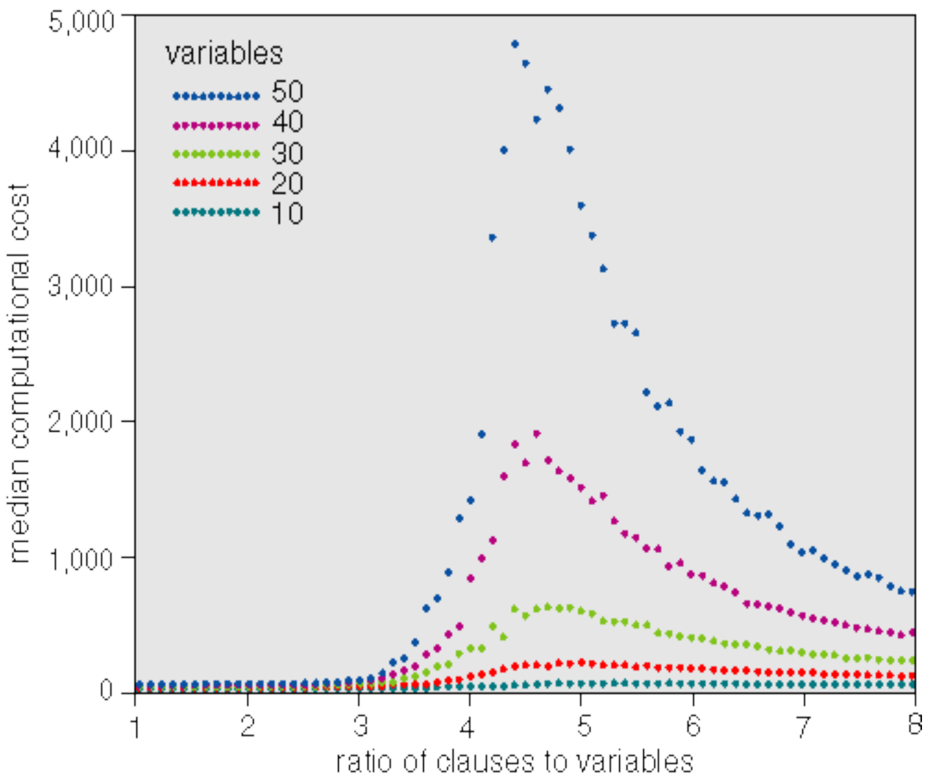


- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - *n* variables, *l* clauses

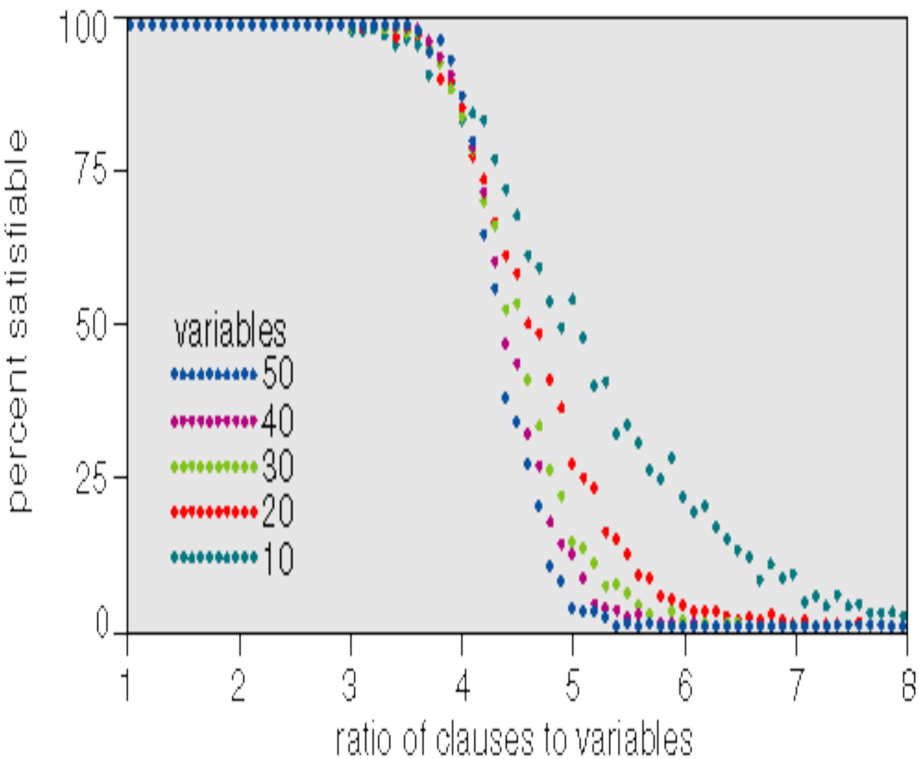
- Which are the hard instances?
  - around *l*/*n* = 4.3

Random 3-SAT

- Varying problem size, *n*
- Complexity peak appears to be largely invariant of algorithm
  - backtracking algorithms like Davis-Putnam
  - local search procedures like GSAT
- *What's so special about 4.3?*



Random 3-SAT



- Complexity peak coincides with solubility transition
  - l/n < 4.3 problems under-constrained and SAT
  - l/n > 4.3 problems over-constrained and UNSAT
  - l/n=4.3, problems on "knife-edge" between SAT and UNSAT

Prop. Logic Themes

- Expressiveness

  Expressive but awkward

  No notion of objects, properties, or relations

  Number of propositions is fixed

- Tractability

  NP in general

  Completeness / speed tradeoff

  Horn clauses, binary clauses