

# Methods of Proof

## Chapter 7, Part II

# Proof methods

- Proof methods divide into (roughly) two kinds:

Application of inference rules:

Legitimate (sound) generation of new sentences from old.

- Resolution
- Forward & Backward chaining

Model checking

Searching through truth assignments.

- Improved backtracking: Davis--Putnam-Logemann-Loveland (DPLL)
- Heuristic search in model space: Walksat.

# Normal Form

We like to prove:  $KB \models \alpha$   
equivalent to:  $KB \wedge \neg\alpha$  unsatisfiable

We first rewrite  $KB \wedge \neg\alpha$  into conjunctive normal form (CNF).

A “conjunction of disjunctions”

$(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

literals

Clause      Clause

- Any KB can be converted into CNF.
- In fact, any KB can be converted into CNF-3 using clauses with at most 3 literals.

# Example: 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:  $\neg(\alpha \vee \beta) = \neg\alpha \wedge \neg\beta$   
 $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply distributive 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

- Resolution: inference rule for CNF: **sound and complete!**

$(A \vee B \vee C)$

$(\neg A)$

“If A or B or C is true, but not A, then B or C must be true.”

-----

$\therefore (B \vee C)$

$(A \vee B \vee C)$

$(\neg A \vee D \vee E)$

“If A is false then B or C must be true, or if A is true then D or E must be true, hence since A is either true or false, B or C or D or E must be true.”

-----

$\therefore (B \vee C \vee D \vee E)$

$(A \vee B)$

$(\neg A \vee B)$

-----

$\therefore (B \vee B) \equiv B$

← Simplification

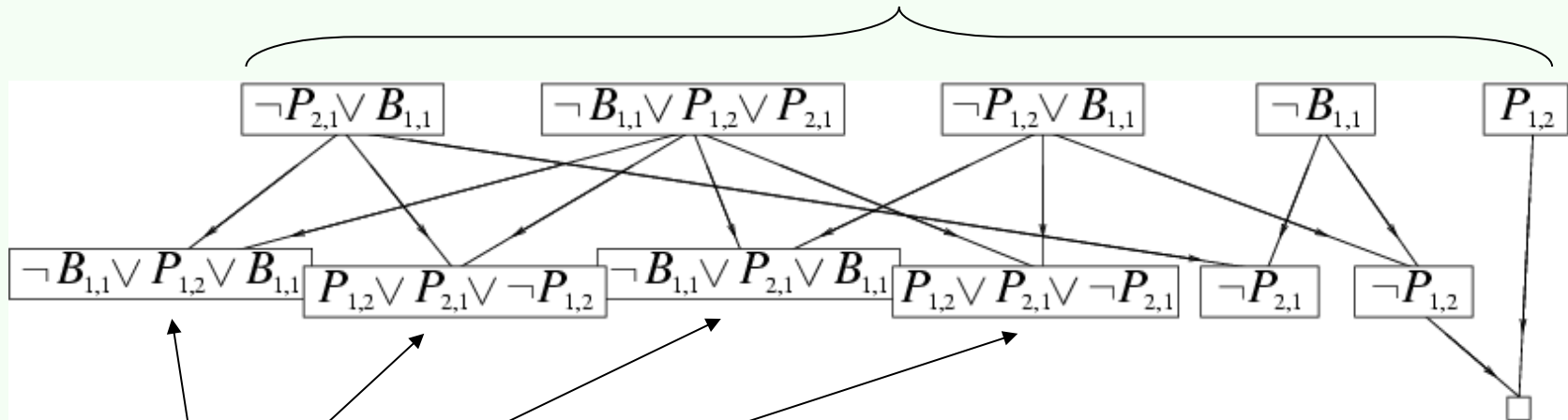
# Resolution Algorithm

- The resolution algorithm tries to prove:  $KB \models \alpha$  equivalent to  $KB \wedge \neg \alpha$  unsatisfiable
- Generate all new sentences from KB and the query.
- One of two things can happen:
  1. We find  $P \wedge \neg P$  which is unsatisfiable. I.e. we can entail the query.
  2. We find no contradiction: there is a model that satisfies the sentence  $KB \wedge \neg \alpha$  (non-trivial) and hence we cannot entail the query.

# Resolution example

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$

$$KB \wedge \neg \alpha$$



True!

False in  
all worlds

# Horn Clauses

- Resolution can be exponential in space and time.
- If we can reduce all clauses to “Horn clauses” resolution is linear in space and time

A clause with at most 1 positive literal.

e.g.  $A \vee \neg B \vee \neg C$

- Every Horn clause can be rewritten as an implication with a conjunction of positive literals in the premises and a single positive literal as a conclusion.

e.g.  $B \wedge C \Rightarrow A$

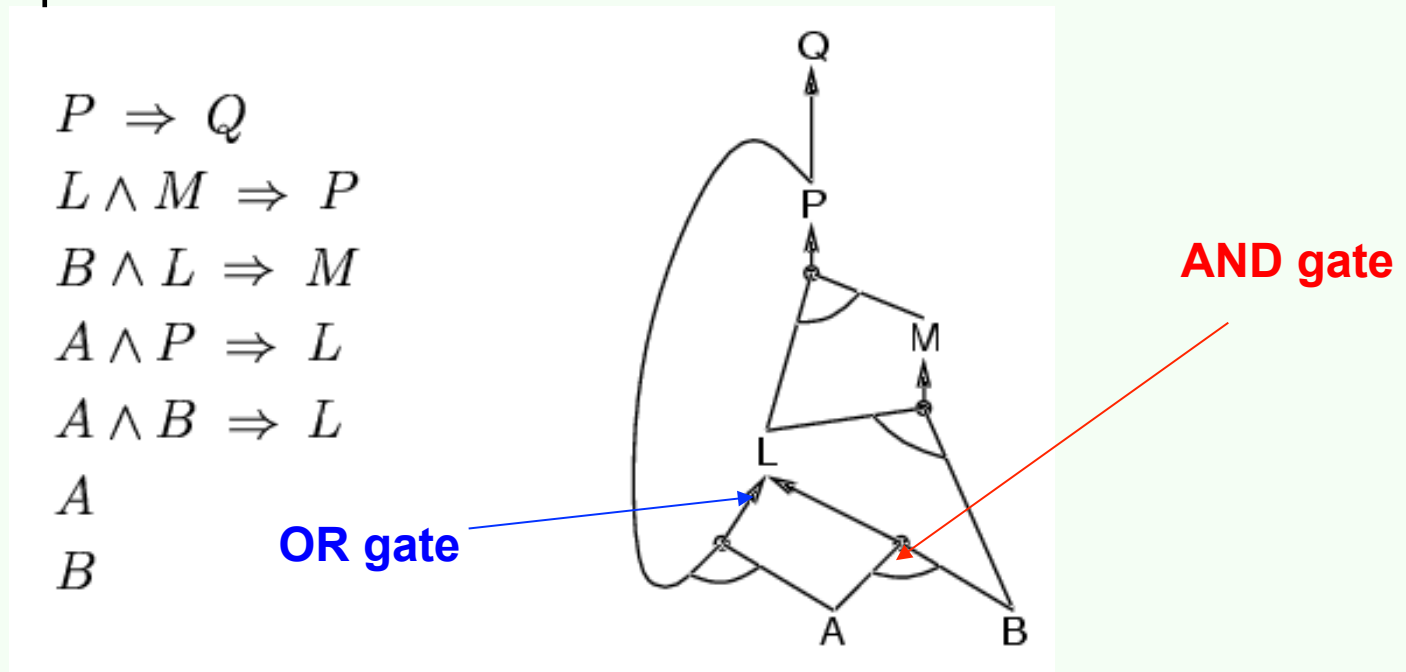
- 1 positive literal: definite clause
- 0 positive literals: Fact or integrity constraint:  
e.g.  $(\neg A \vee \neg B) \equiv (A \wedge B \Rightarrow \text{False})$
- Forward Chaining and Backward chaining are sound and complete with Horn clauses and run linear in space and time.

# Try it Yourself

- 7.9 page 238: (Adapted from Barwise and Etchemendy (1993).) If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.
- *Derive the KB in normal form.*
- *Prove: Horned, Prove: Magical.*

# Forward chaining

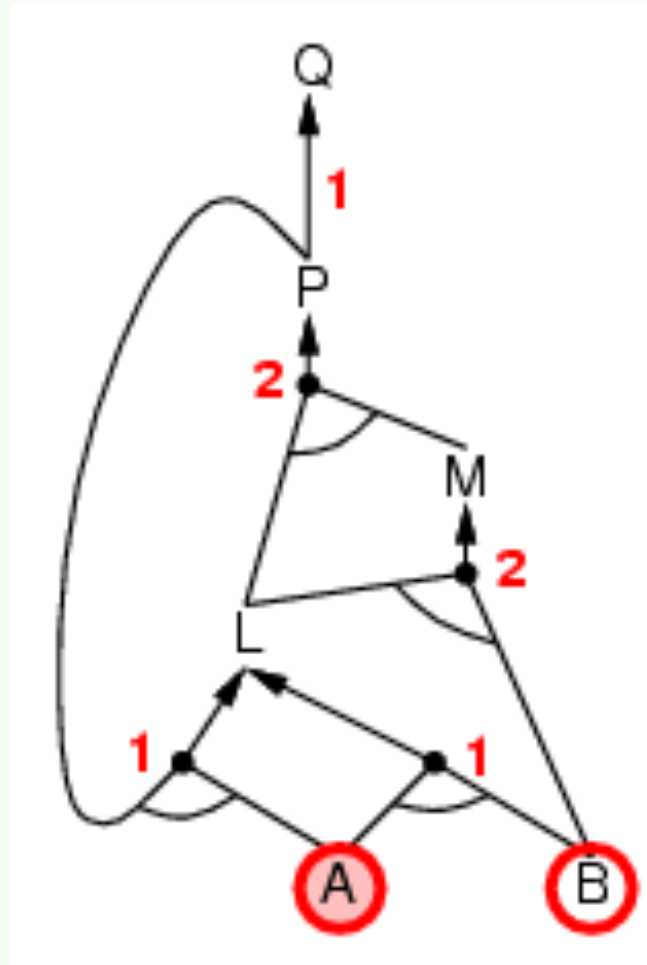
- Idea: fire any rule whose premises are satisfied in the *KB*, add its conclusion to the *KB*, until query is found.
- This proves that  $KB \Rightarrow Q$  is true in all possible worlds (i.e. trivial), and hence it proves entailment.



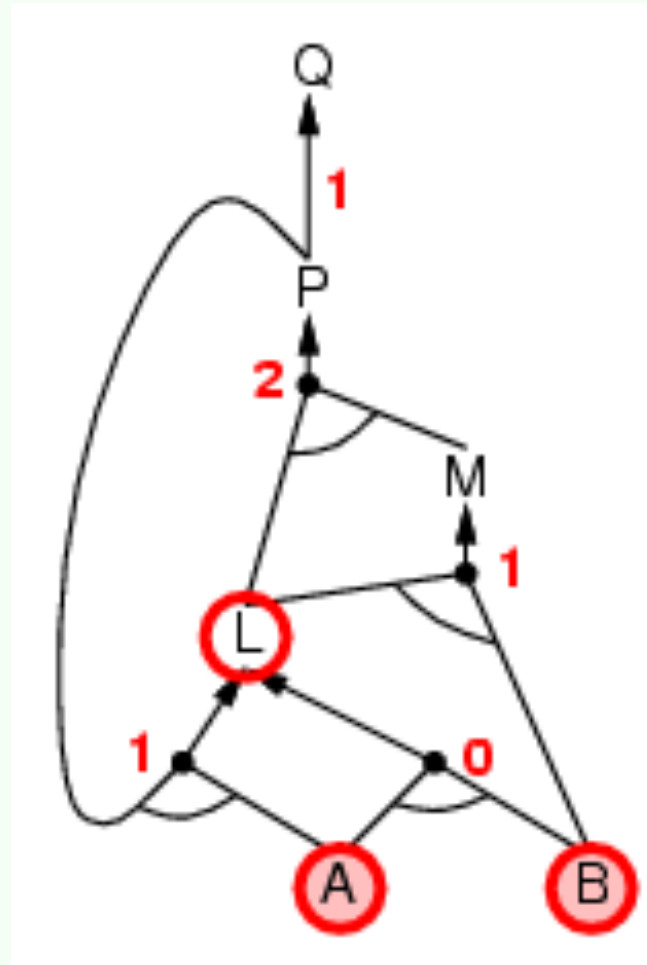
- Forward chaining is sound and complete for Horn KB



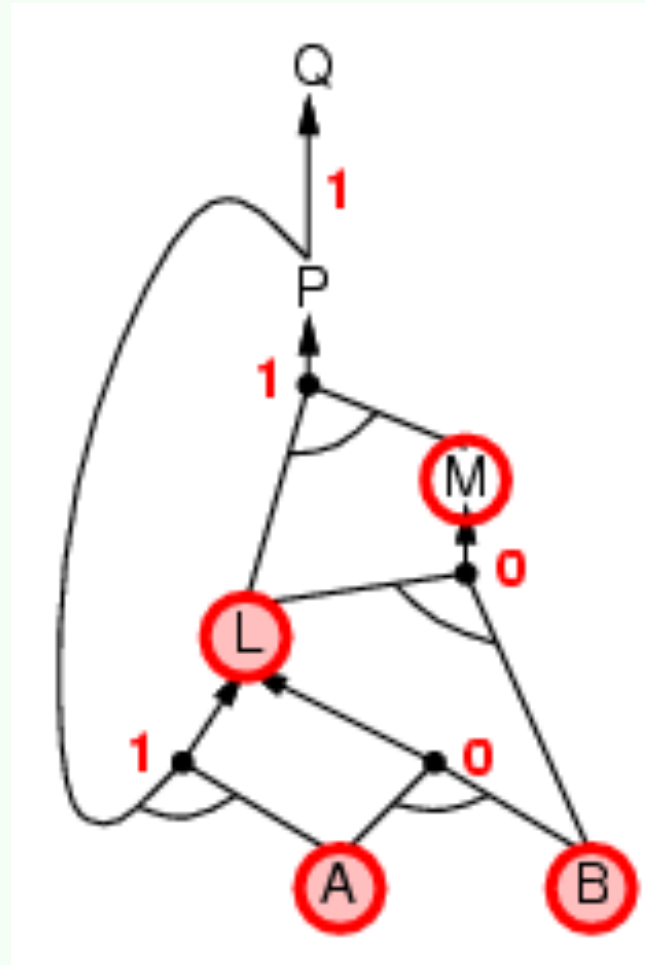
# Forward chaining example



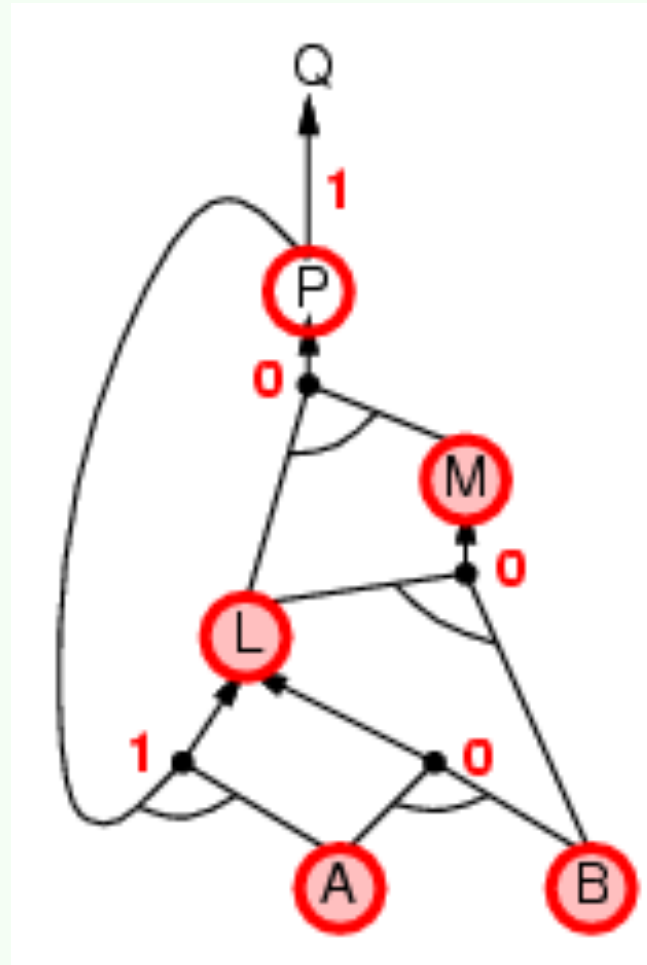
# Forward chaining example



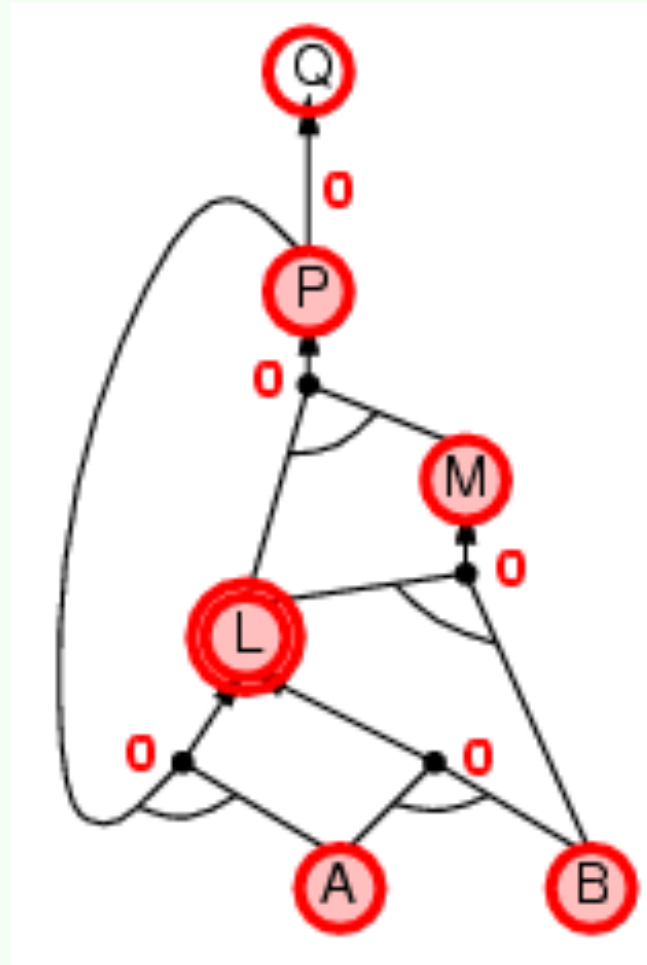
# Forward chaining example



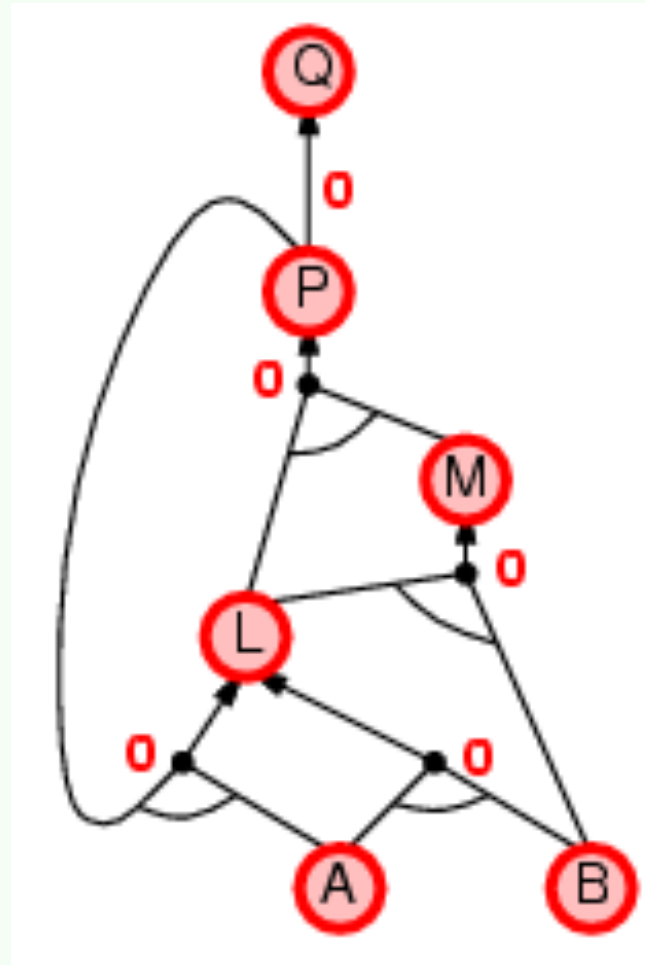
# Forward chaining example



# Forward chaining example



# Forward chaining example



# Backward chaining

Idea: work backwards from the query  $q$

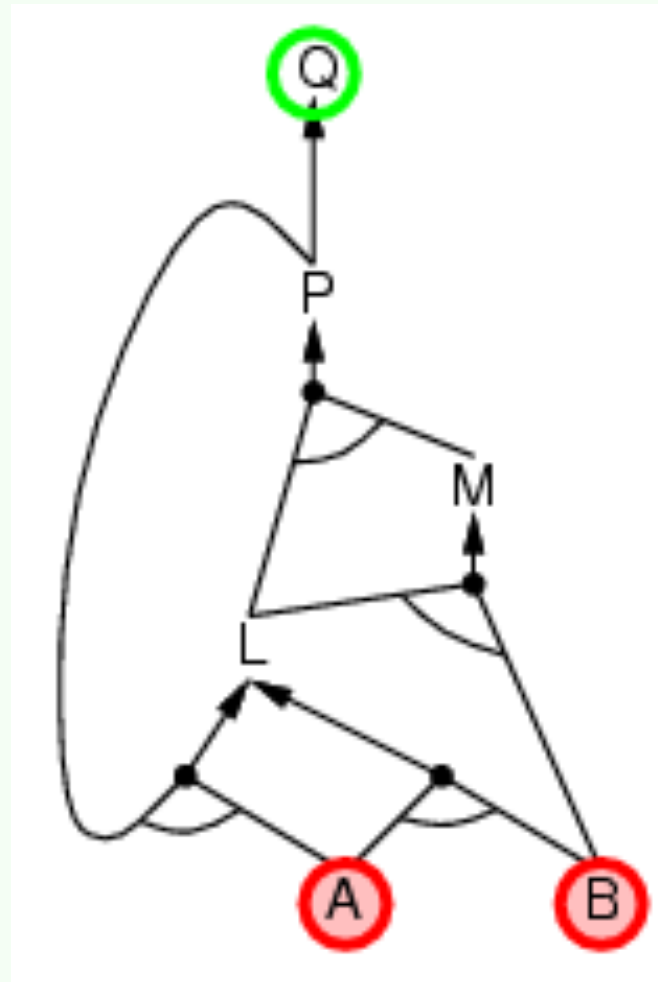
- check if  $q$  is known already, or
- prove by BC all premises of some rule concluding  $q$
- Hence BC maintains a stack of sub-goals that need to be proved to get to  $q$ .

Avoid loops: check if new sub-goal is already on the goal stack

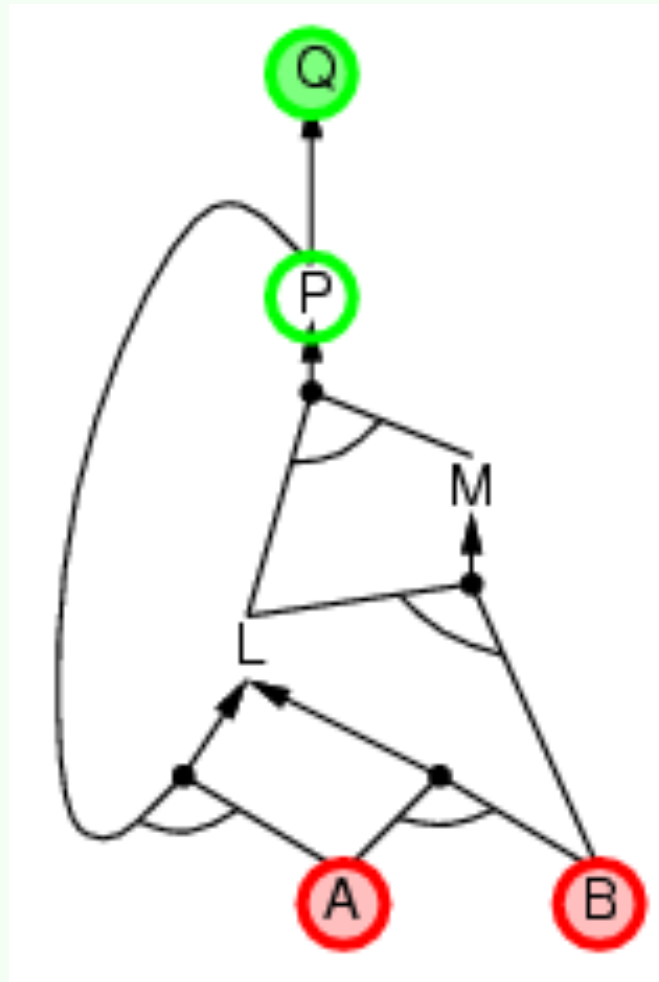
Avoid repeated work: check if new sub-goal

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

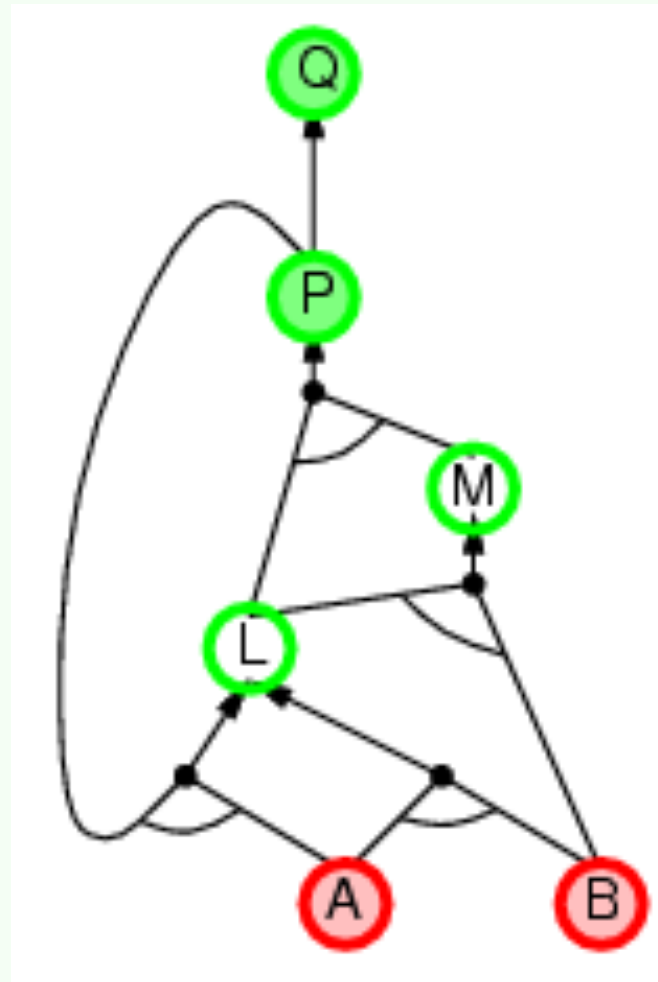
# Backward chaining example



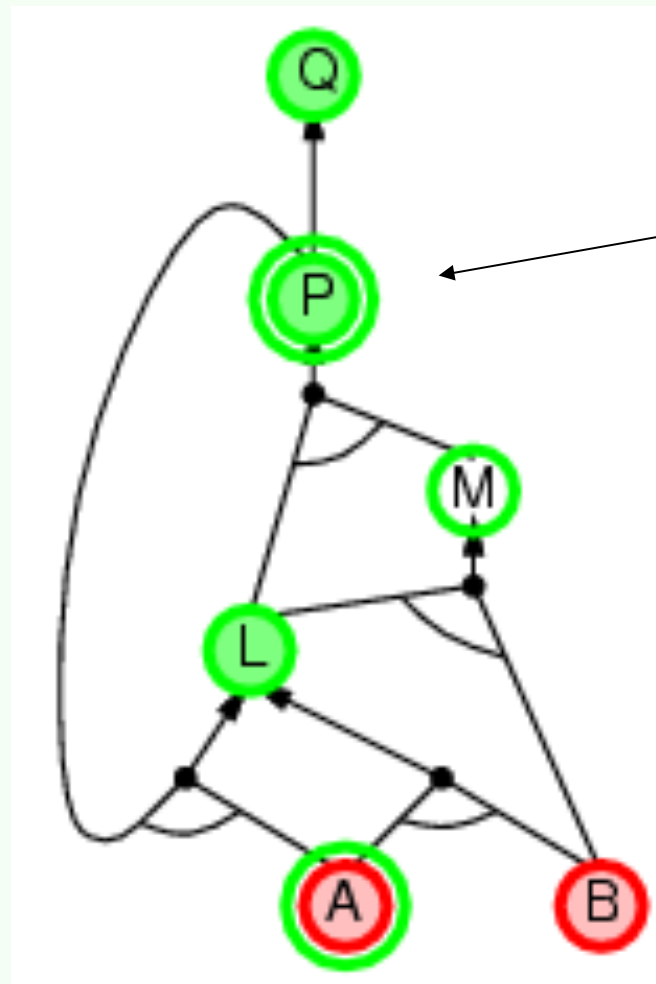
# Backward chaining example



# Backward chaining example

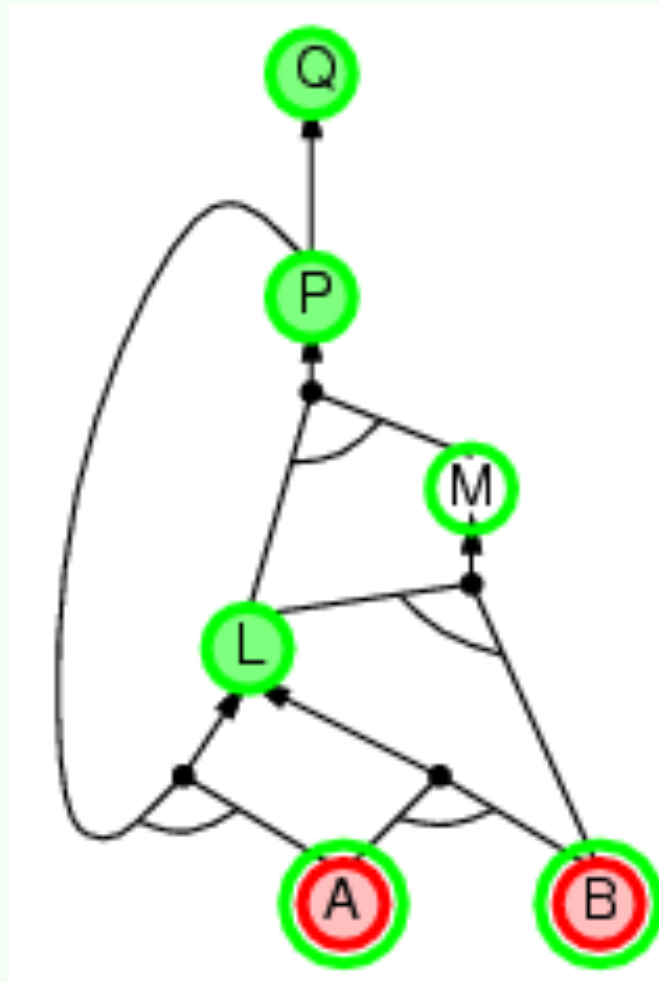


# Backward chaining example



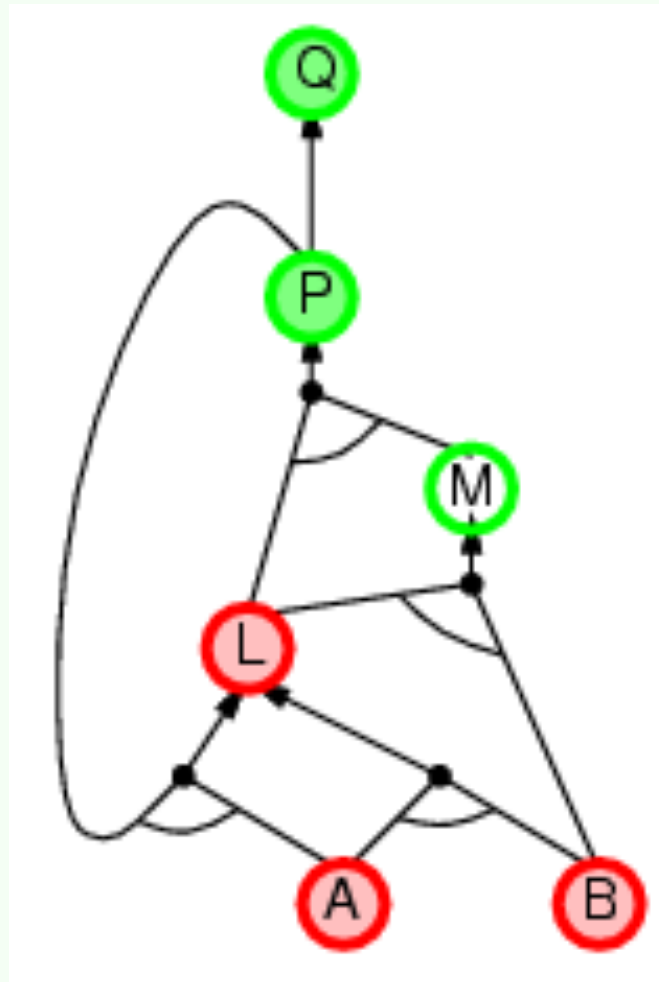
we need P to prove L and L to prove P.

# Backward chaining example

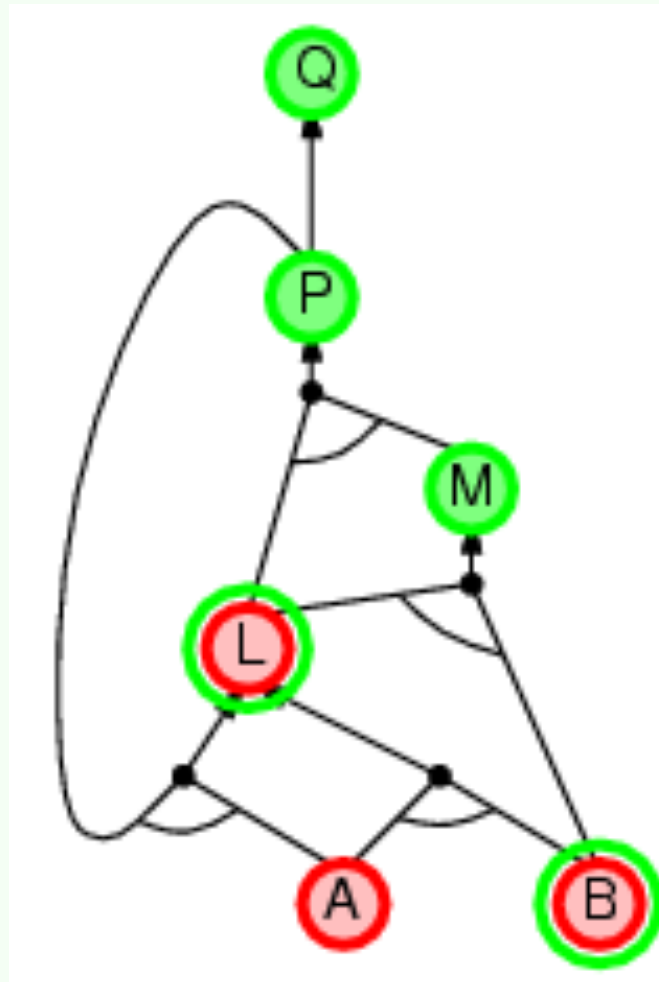


As soon as you can move forward, do so.

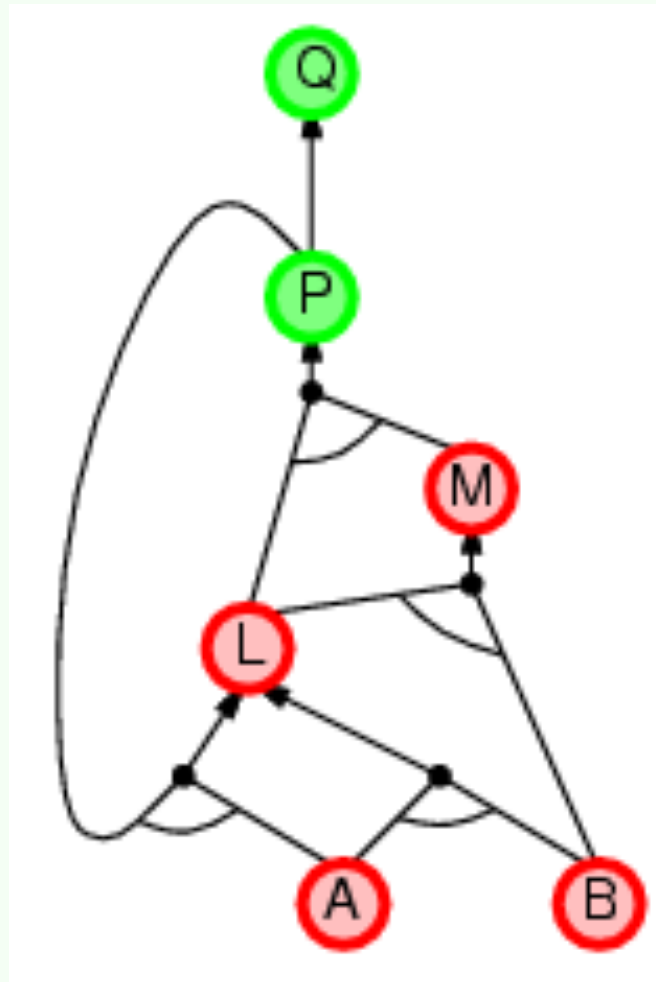
# Backward chaining example



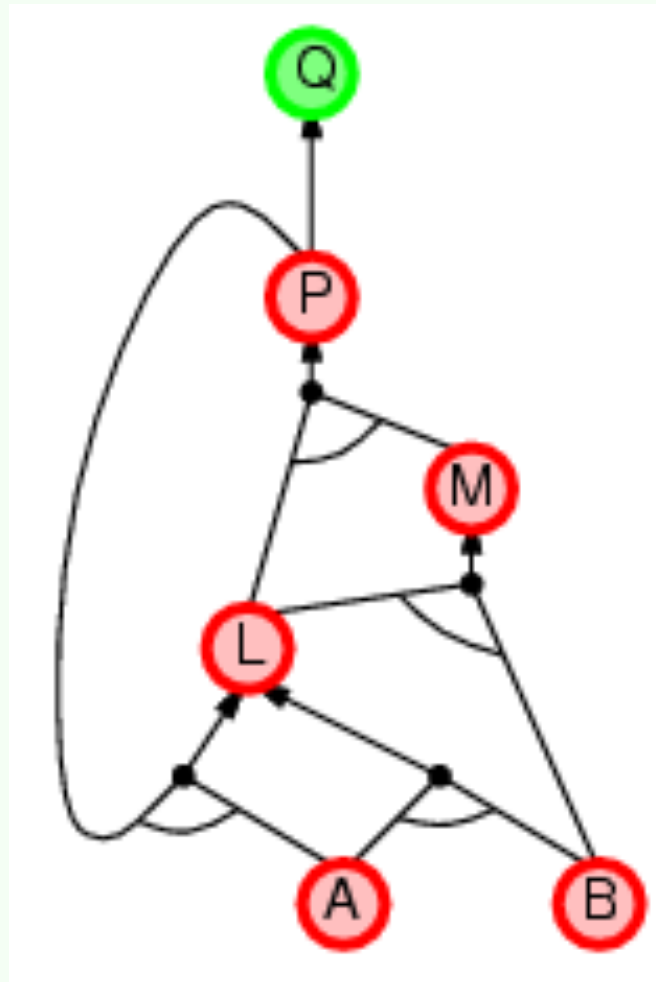
# Backward chaining example



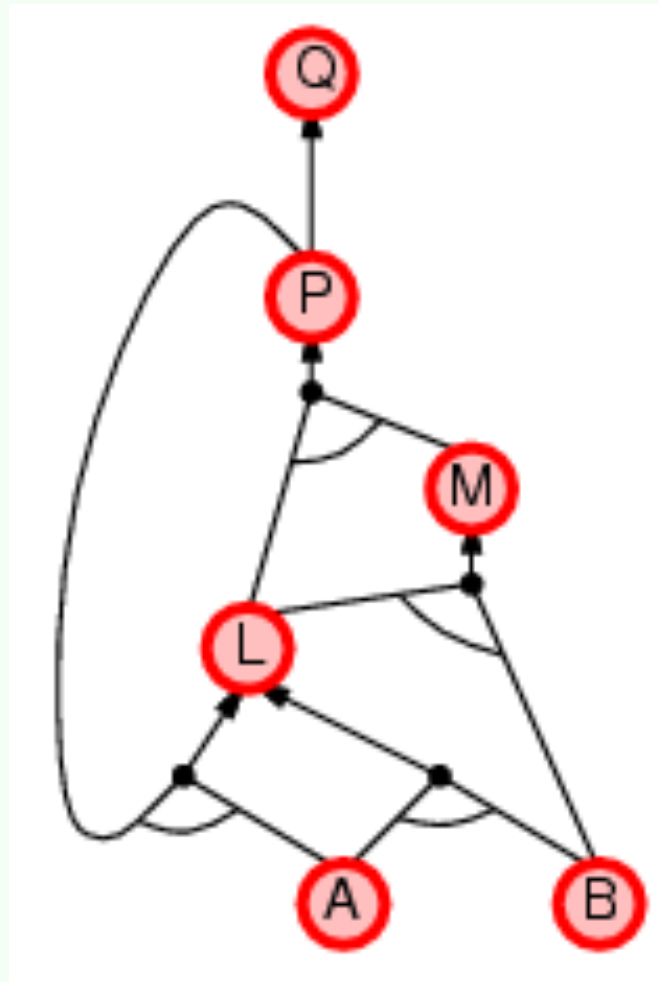
# Backward chaining example



# Backward chaining example



# Backward chaining example



# Forward vs. backward chaining

- FC is **data-driven**, automatic, unconscious processing,
  - e.g., object recognition, routine decisions
- May do lots of work that is irrelevant to the goal
- BC is **goal-driven**, appropriate for problem-solving,
  - e.g., Where are my keys? How do I get into a PhD program?
- Complexity of BC can be **much less** than linear in size of KB

# Model Checking

Two families of efficient algorithms:

- Complete backtracking search algorithms: DPLL algorithm
- Incomplete local search algorithms
  - WalkSAT algorithm

# The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable. **This is just backtracking search for a CSP.**

Improvements:

1. Early termination

A clause is true if any literal is true.

A sentence is false if any clause is false.

2. Pure symbol heuristic

Pure symbol: always appears with the same "sign" in all clauses.

e.g., In the three clauses  $(A \vee \neg B)$ ,  $(\neg B \vee \neg C)$ ,  $(C \vee A)$ , A and B are pure, C is impure.

Make a pure symbol literal true. (if there is a model for S, then making a pure symbol true is also a model).

3 Unit clause heuristic

Unit clause: only one literal in the clause

The only literal in a unit clause must be true.

Note: literals can become a pure symbol or a unit clause when other literals obtain truth values. e.g.

$$(\cancel{A \vee True}) \wedge (\neg A \vee B)$$

$$A = \text{pure}$$

# The WalkSAT algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness

## Walksat Procedure

Start with random initial assignment.

Pick a random unsatisfied clause.

Select and flip a variable from that clause:

With probability  $p$ , pick a **random** variable.

With probability  $1-p$ , pick **greedily**

a variable that minimizes the number of unsatisfied clauses

Repeat to predefined maximum number flips;  
if no solution found, restart.

# Hard satisfiability problems

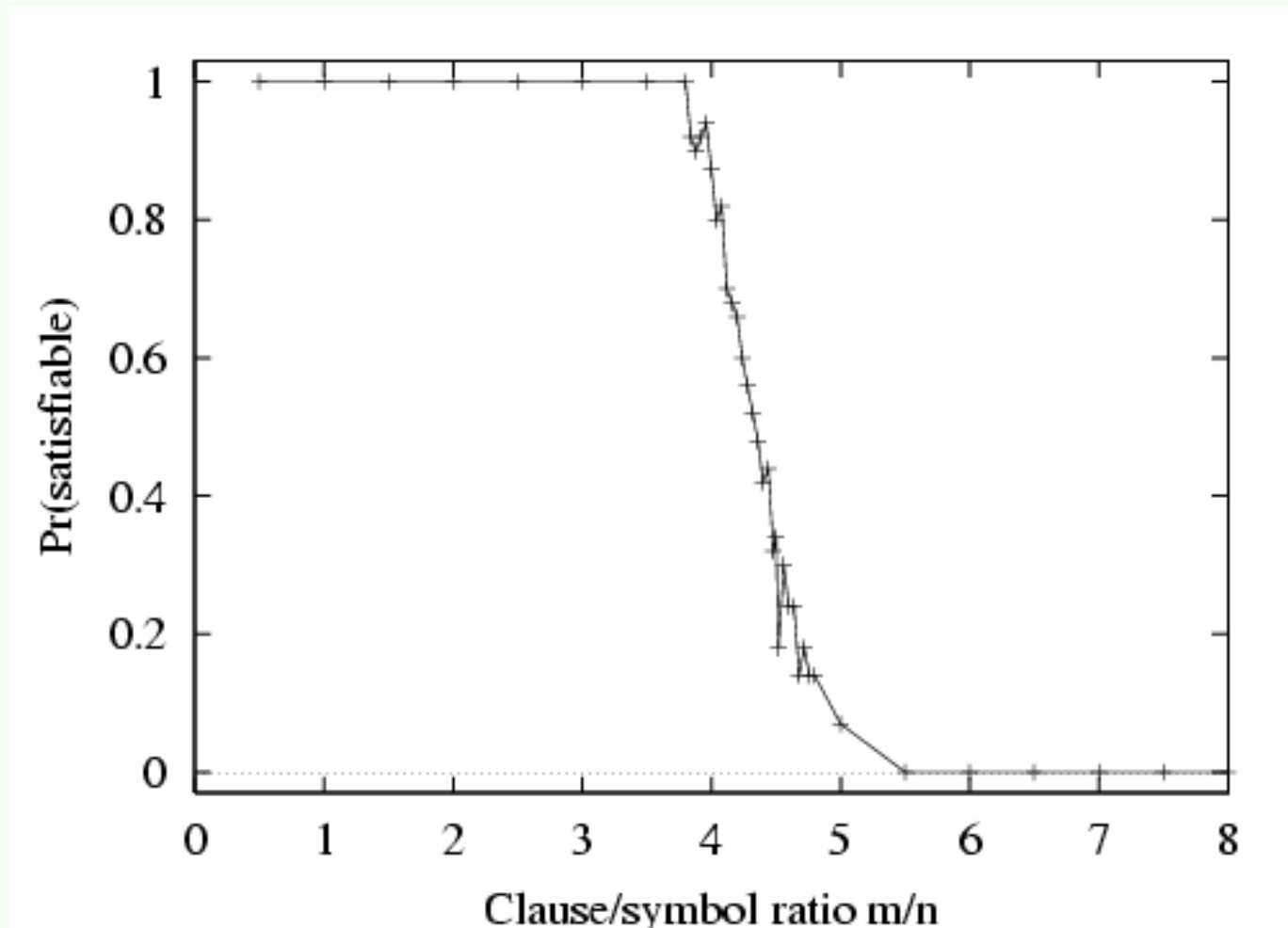
- Consider *random* 3-CNF sentences. e.g.,  
 $(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$

$m$  = number of clauses (5)

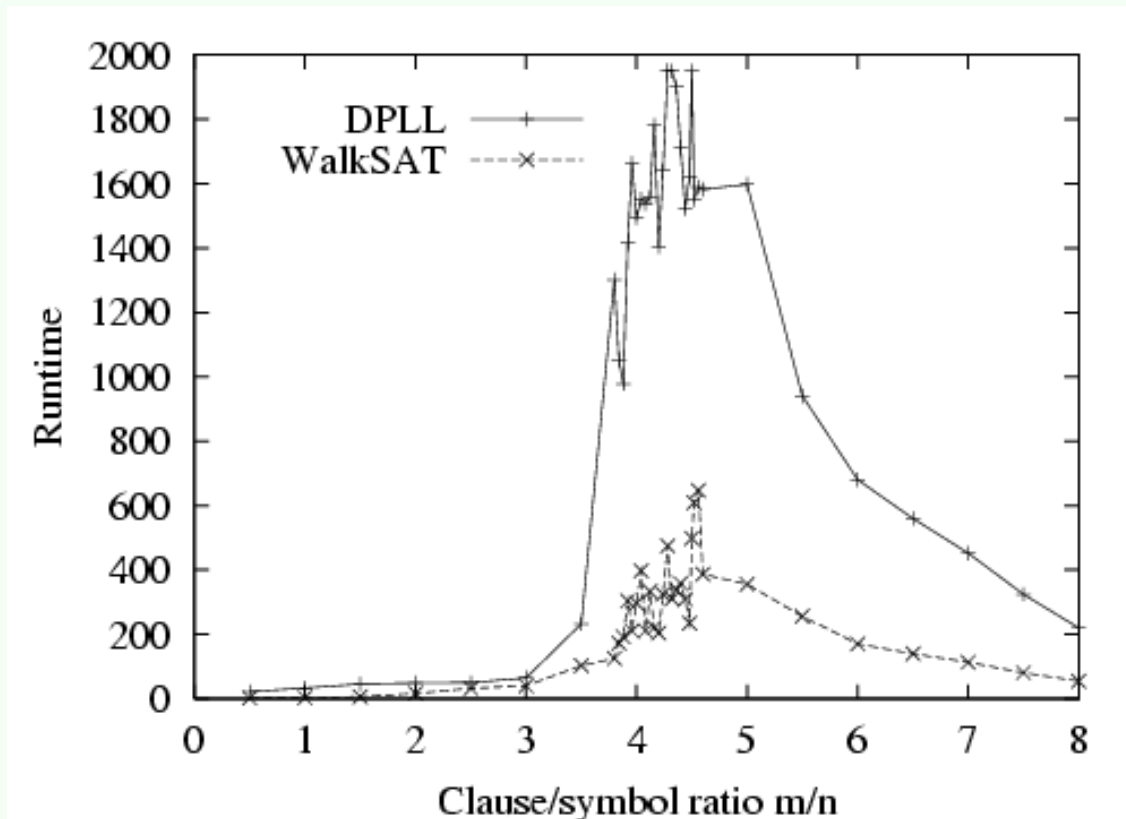
$n$  = number of symbols (5)

- Hard problems seem to cluster near  $m/n = 4.3$   
(critical point)

# Hard satisfiability problems



# Hard satisfiability problems



- Median runtime for 100 **satisfiable** random 3-CNF sentences,  $n = 50$

# Summary

- Logical agents apply **inference** to a **knowledge base** to derive new information and make decisions
- Basic concepts of logic:
  - **syntax**: formal structure of **sentences**
  - **semantics**: **truth** of sentences wrt **models**
  - **entailment**: necessary truth of one sentence given another
  - **inference**: deriving sentences from other sentences
  - **soundness**: derivations produce only entailed sentences
  - **completeness**: derivations can produce all entailed sentences
- Resolution is complete for propositional logic  
Forward, backward chaining are linear-time, complete for Horn clauses
- Propositional logic lacks expressive power