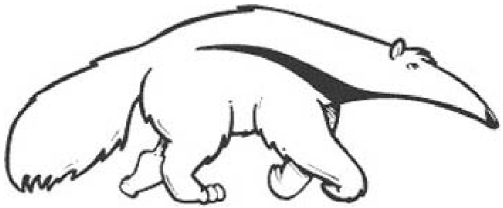
Propositional Logic

CS171, Fall 2011

Introduction to Artificial Intelligence

Prof. Alexander Ihler

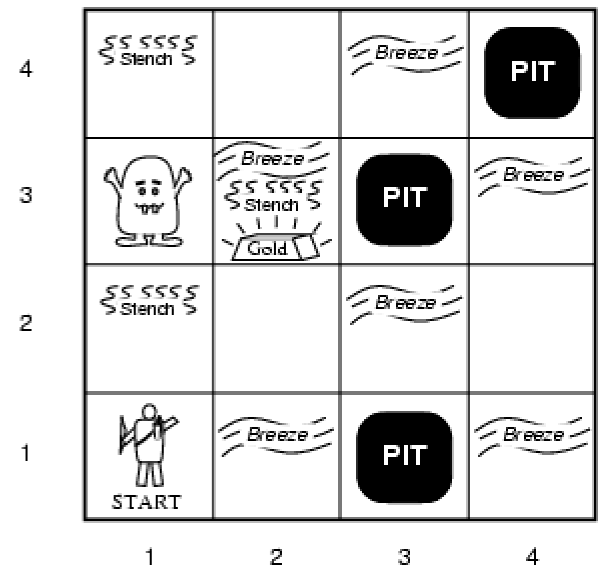


BREN:ICS
INFORMATION AND COMPUTER SCIENCES

UNIVERSITY of CALIFORNIA IRVINE

Wumpus World test-bed

- Performance measure
  - gold +1000, death -1000
  - -1 per step, -10 for using the arrow
- Environment
  - Squares adjacent to wumpus are smelly
  - Squares adjacent to pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square

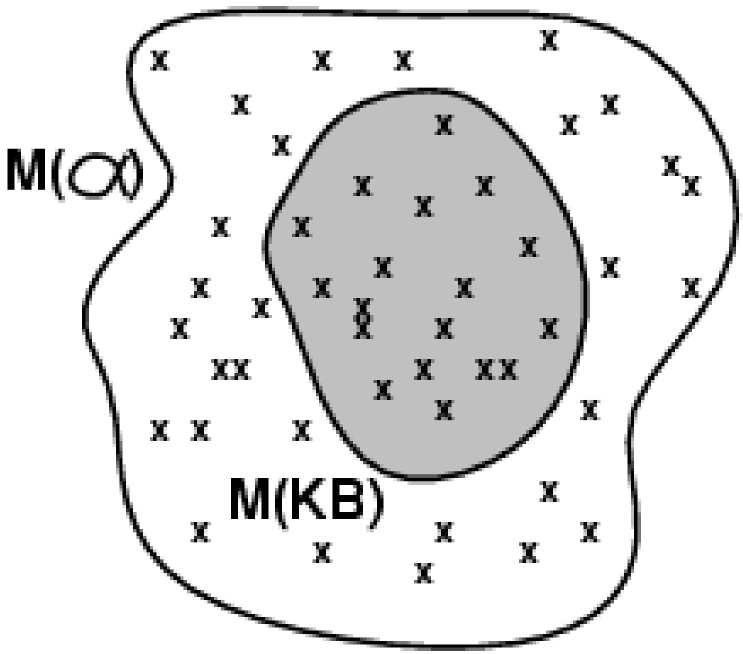


- Sensors: Stench, Breeze, Glitter, Bump, Scream
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot

Models

- Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

- We say m is a model of a sentence α if α is true in m
- M(α) is the set of all models of α



- Then KB $\models$ α iff $M(KB) \subseteq M(\alpha)$

  – E.g. KB = Giants won and Reds won α = Giants won

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.

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 algorithm
    - Typically require transformation of sentences into a normal form
  - Model checking
    - truth table enumeration (always exponential in n)
    - improved backtracking, e.g., Davis--Putnam-Logemann-Loveland (DPLL), Backtracking with constraint propagation, backjumping.
    - heuristic search in model space (sound but incomplete)
    - e.g., min-conflicts-like hill-climbing algorithms

Resolution in Propositional Calculus

- **Using clauses as wffs**
  - Literal, clauses, conjunction of clauses (cnfs)
- **Resolution rule:** $(A \vee B \vee C) \wedge \neg B \models (A \vee C)$
- **Resolving (P V Q) and (P V $\neg$ Q) $\vdash$ P**
  - Generalize modus ponens, chaining .
  - Resolving a literal with its negation yields empty clause.
- **Resolution is sound**
- **Resolution is NOT complete:**
  - P and R entails P V R but you cannot infer P V R From (P and R) by resolution
- **Resolution is complete for refutation:** adding ($\neg$P) and ($\neg$R) to (P and R) we can infer the empty clause.
- **Decidability of propositional calculus by resolution refutation:** if a sentence w is not entailed by KB then resolution refutation will terminate without generating the empty clause.

Resolution

Conjunctive Normal Form (CNF—universal)

conjunction of $\underbrace{\textit{disjunctions} \text{ of } \textit{literals}}_{\textit{clauses}}$

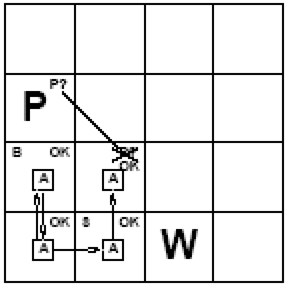
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where ℓ_i and m_j are complementary literals. E.g.,

$$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$



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 ($\wedge$ over $\vee$) and flatten:

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

Ex: converting to conjunctive clauses

- **1. Eliminate implications**

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

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

- **2. Reduce the scope of negation sign**

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

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

- **3. Convert to cnfs using the associative and distributive laws**

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

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

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

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

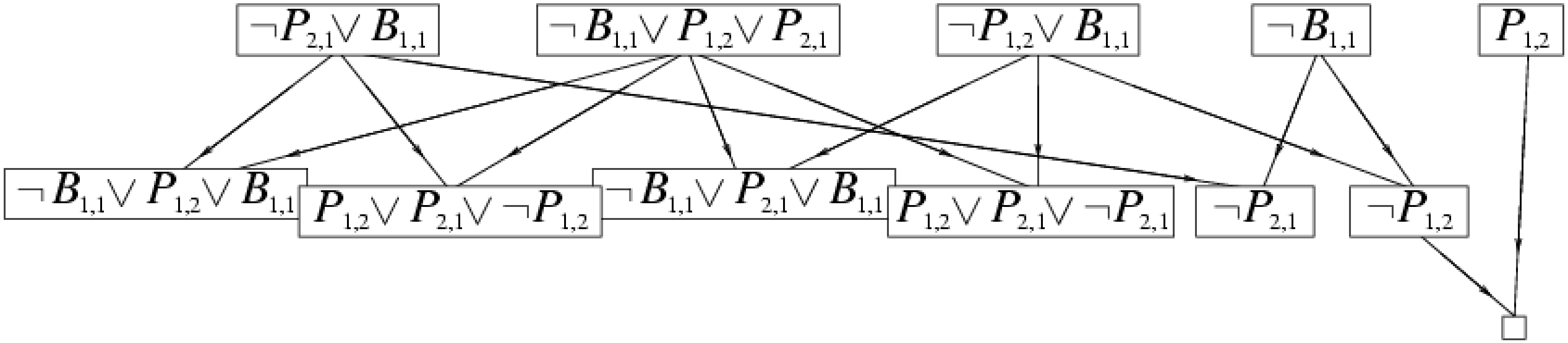
Resolution algorithm

- Proof by contradiction, i.e., show *KB*∧¬α 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 Ci, Cj in clauses do
            resolvents ← PL-RESOLVE(Ci, Cj)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Resolution example

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



Soundness of resolution

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
False	*False*	*False*	*False*	*True*	*False*
False	*False*	*True*	*False*	*True*	*True*
False	*True*	*False*	*True*	*False*	*False*
<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
<u>*True*</u>	<u>*False*</u>	<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
<u>*True*</u>	<u>*False*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>
True	*True*	*False*	*True*	*False*	*True*
<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>	<u>*True*</u>

Figure 6.14 A truth table demonstrating the soundness of the resolution inference rule. We have underlined the rows where both premises are true.

The party example

- **If Alex goes, then Beki goes: $A \rightarrow B$**
- **If Chris goes, then Alex goes: $C \rightarrow A$**
- **Beki does not go: not B**
- **Chris goes: C**
- **Query: Is it possible to satisfy all these conditions?**

- **Should I go to the party?**

Example of proof by Refutation

- **Assume the claim is false and prove inconsistency:**
  - Example: can we prove that Chris will not come to the party?
- **Prove by generating the desired goal.**

$A \rightarrow B, \neg B$

$C \rightarrow A$

- **Prove by refutation: add the negation of the goal and prove no model**
- **Proof:**

$$\text{from } A \rightarrow B, \neg B \text{ infer } \neg A$$

$$\text{from } C \rightarrow A, \neg A \text{ infer } \neg C$$

- **Refutation:**

$$\begin{tikzcd}
A \rightarrow B \arrow[dr] & & \neg B \arrow[dl] & & C \rightarrow A \arrow[dl] & & \neg(\neg C) \arrow[dddl] \\
& \neg A \arrow[dr] & & & & & \\
& & & \neg C \arrow[drr] & & & \\
& & & & & \varphi &
\end{tikzcd}$$

Proof by refutation

- **Given a database KB in conjunctive normal form**
  - Find a sequence of resolution steps from KB to the empty clauses
    - Use the search space paradigm:
      - <u>States:</u> current cnf KB + new clauses
      - <u>Operators:</u> resolution
      - <u>Initial state:</u> KB + negated goal
      - <u>Goal State:</u> a database containing the empty clause
      - Search using any search method

Proof by refutation (contd.)

- **Or:**
  - Prove that KB has no model - PSAT
    - A cnf theory is a constraint satisfaction problem:
      - variables: the propositions
      - domains: true, false
      - constraints: clauses (or their truth tables)
      - Find a solution to the csp. If no solution no model.
      - This is the satisfiability question
      - Methods: Backtracking arc-consistency ≈ unit resolution, local search

Resolution refutation search strategies

- **Ordering strategies**
  - Breadth-first, depth-first
  - I-level resolvents are generated from level-(I-1) or less resolvents
  - Unit-preference: prefer resolutions with a literal
- **Set of support**:
  - Allows reslutions in which one of the resolvents is in the set of support
  - The set of support: those clauses coming from negation of the theorem or their decendents.
  - The set of support strategy is refutation complete
- **Linear input**
  - Restricted to resolutions when one member is in the input clauses
  - Linear input is not refutation complete
  - Example: (PVQ) (P V not Q) (not P V Q) (not P V not Q) have no model

Complexity of propositional inference

- **Checking truth tables is exponential**
- **Satisfiability is NP-complete**
- **However, frequently generating proofs is easy.**
- **Propositional logic is monotonic**
  - If you can entail alpha from knowledge base KB and if you add sentences to KB, you can infer alpha from the extended knowledge-base as well.
- **Inference is local**
  - Tractable Classes: Horn, 2-SAT
- **Horn theories:**
  - Q <-- $P_1, P_2, \ldots P_n$
  - Pi is an atom in the language, Q can be false.
- **Solved by modus ponens or “unit resolution”.**

Forward and backward chaining

Horn Form (restricted)

KB = *conjunction* of *Horn clauses*

Horn clause =

◇ proposition symbol; or

◇ (conjunction of symbols) $\Rightarrow$ symbol

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

Modus Ponens (for Horn Form): complete for Horn KBs

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

Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in *linear* time

Forward chaining algorithm

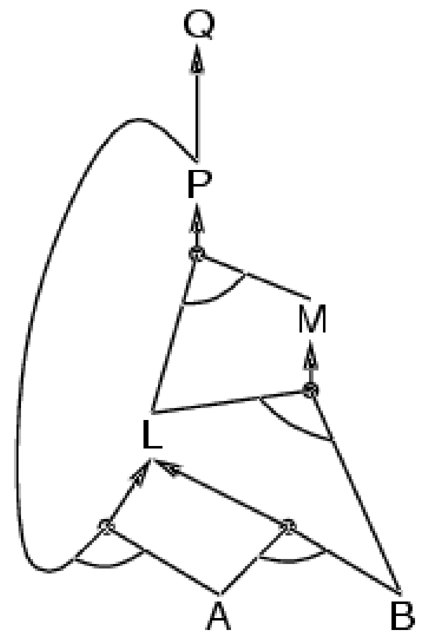
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

Forward chaining

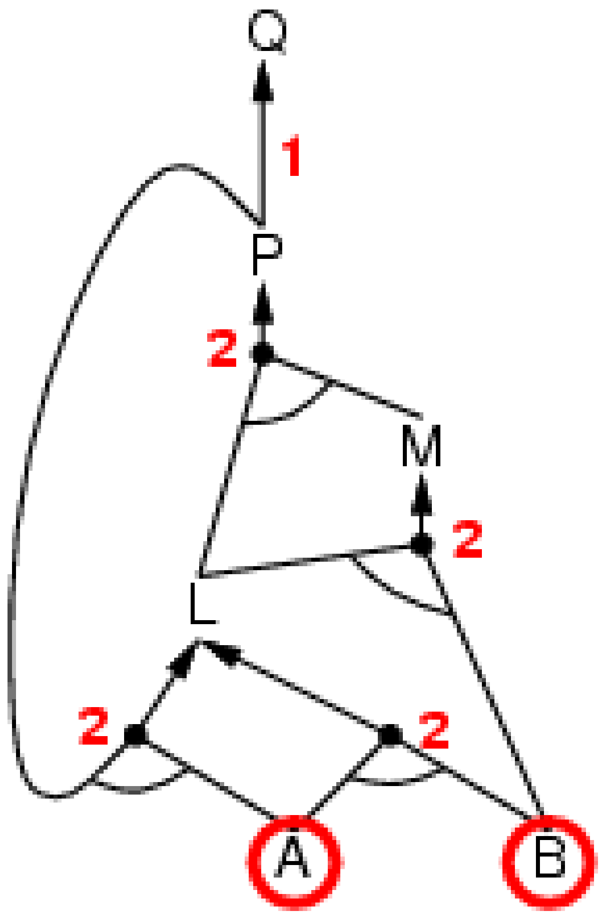
- Idea: fire any rule whose premises are satisfied in the *KB*,
  - add its conclusion to the *KB*, until query is found

$$
\begin{aligned}
&P \Rightarrow Q \\
&L \wedge M \Rightarrow P \\
&B \wedge L \Rightarrow M \\
&A \wedge P \Rightarrow L \\
&A \wedge B \Rightarrow L \\
&A \\
&B
\end{aligned}
$$



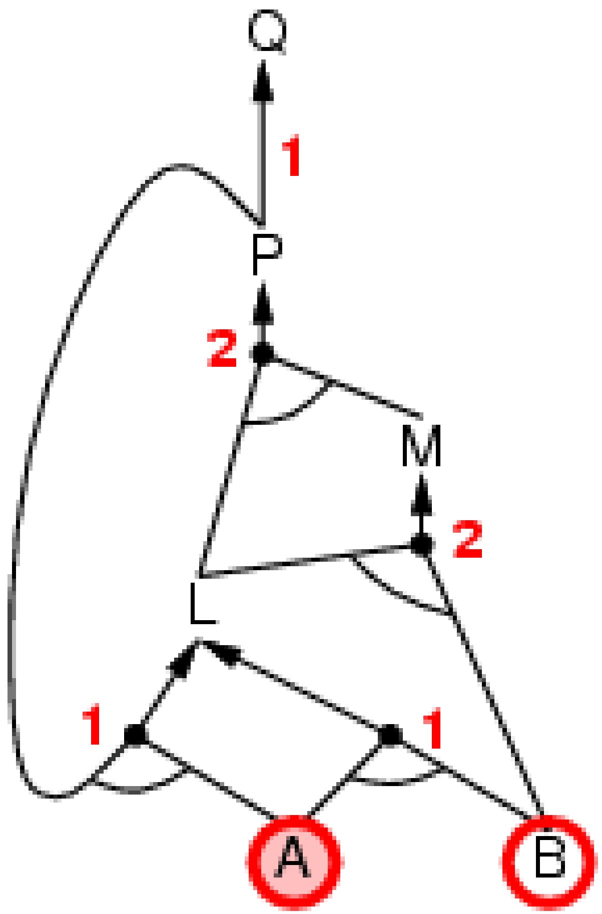
Forward chaining example

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



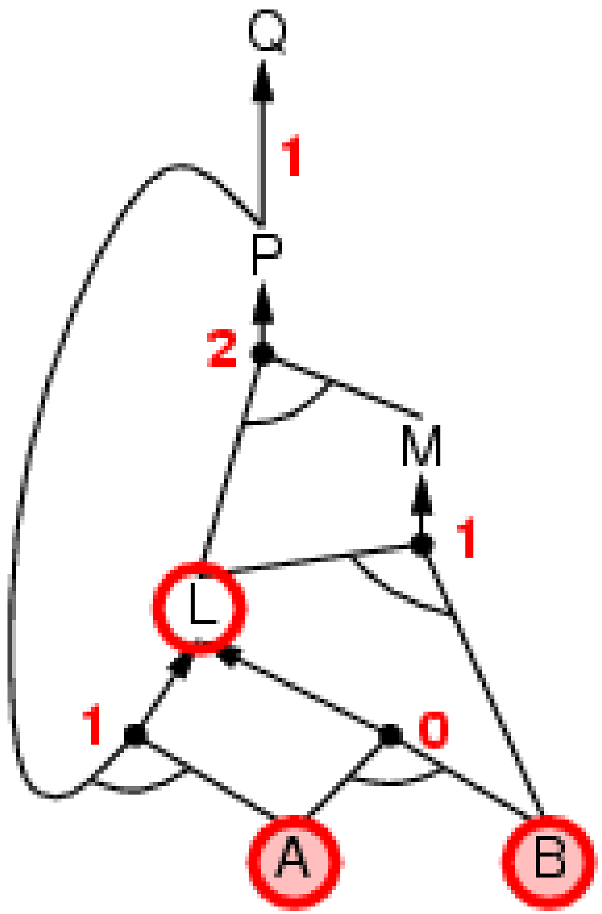
Forward chaining example

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



Forward chaining example

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



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

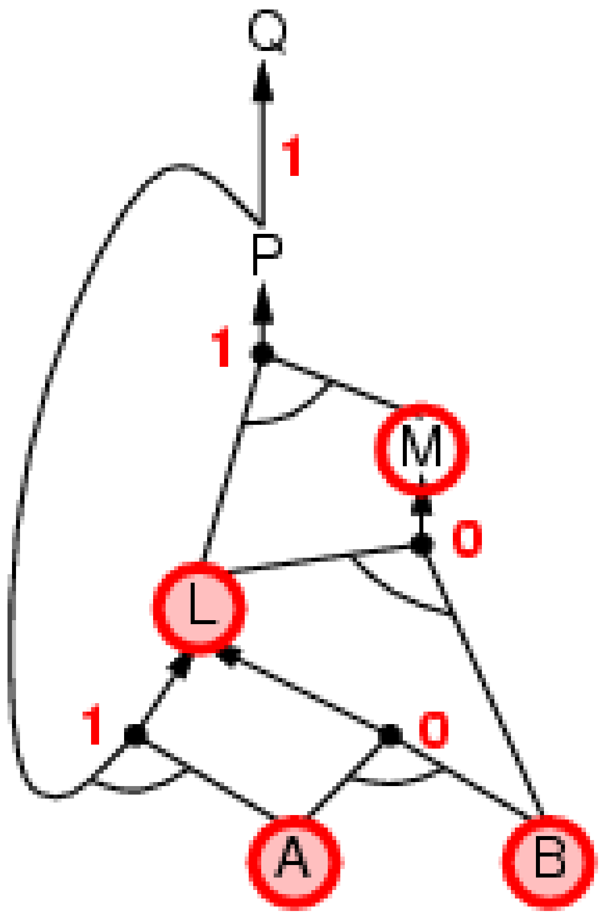
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

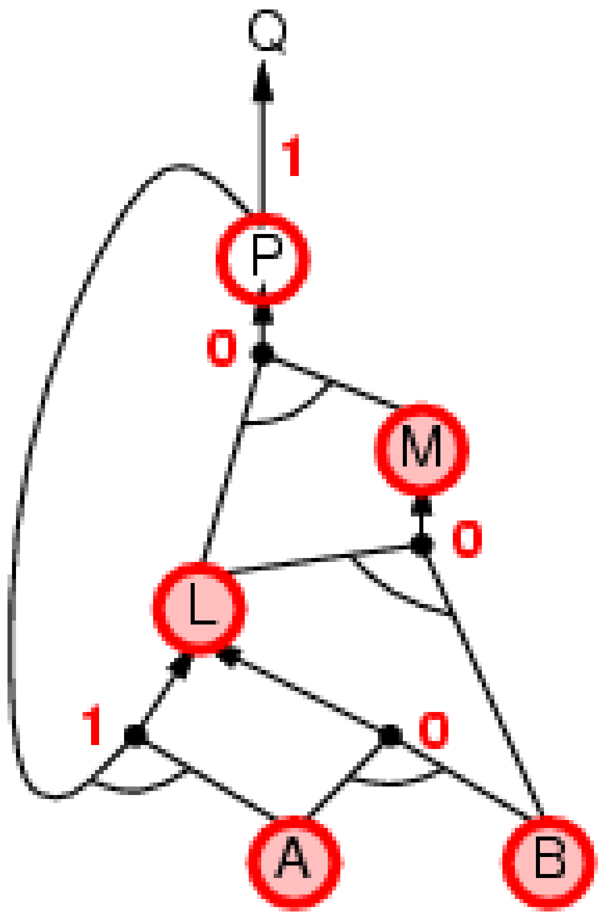
A

B



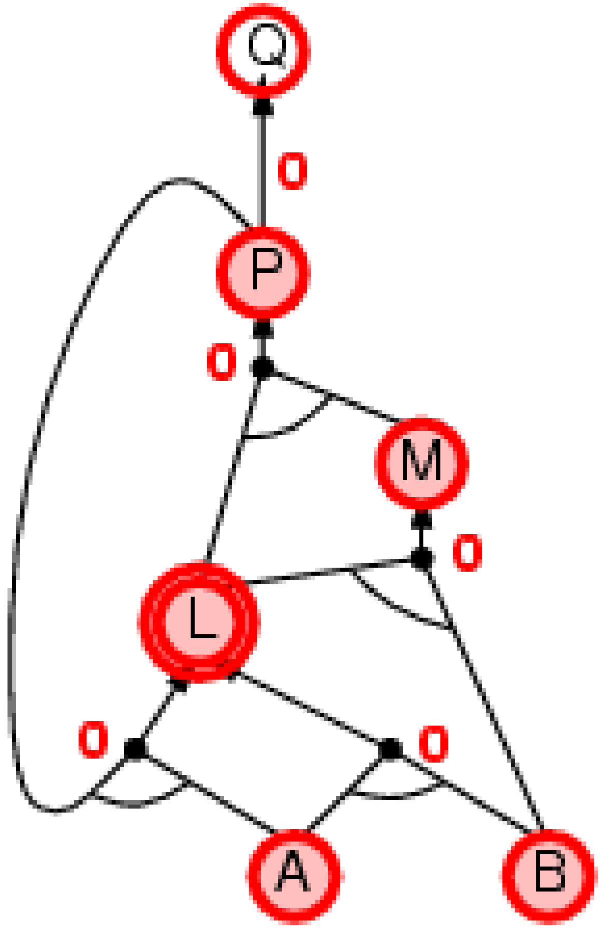
Forward chaining example

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



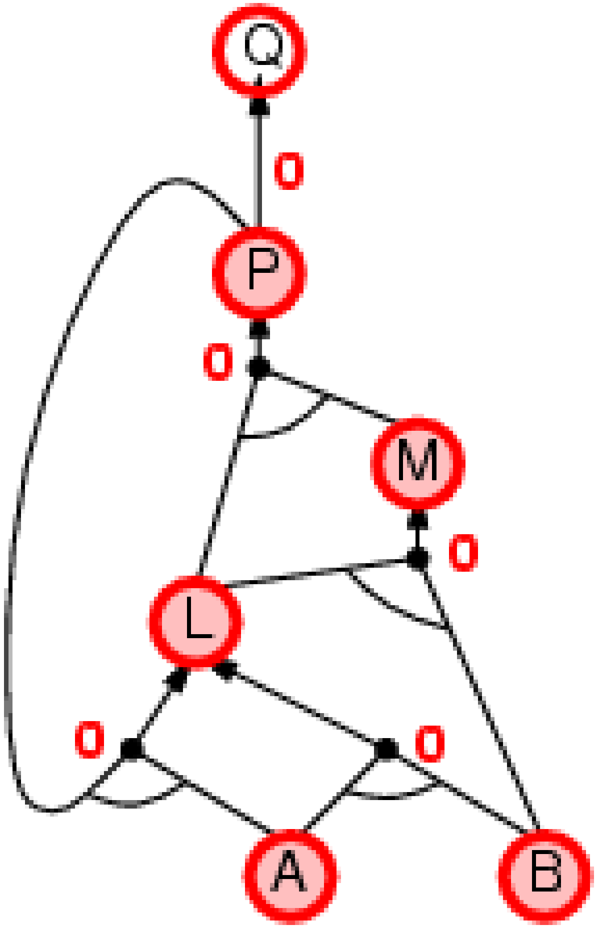
Forward chaining example

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



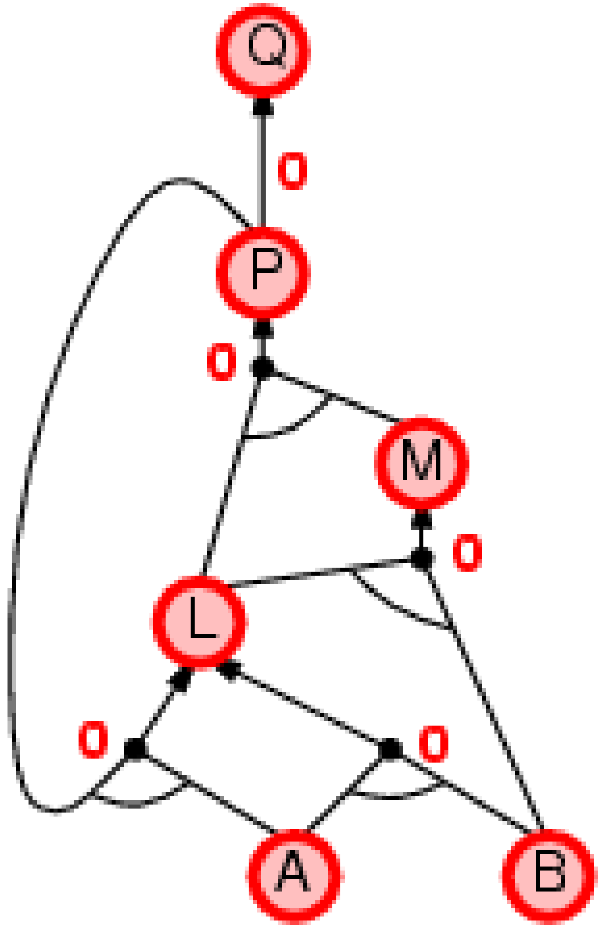
Forward chaining example

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



Forward chaining example

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



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 the goal stack

Avoid repeated work: check if new subgoal

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

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

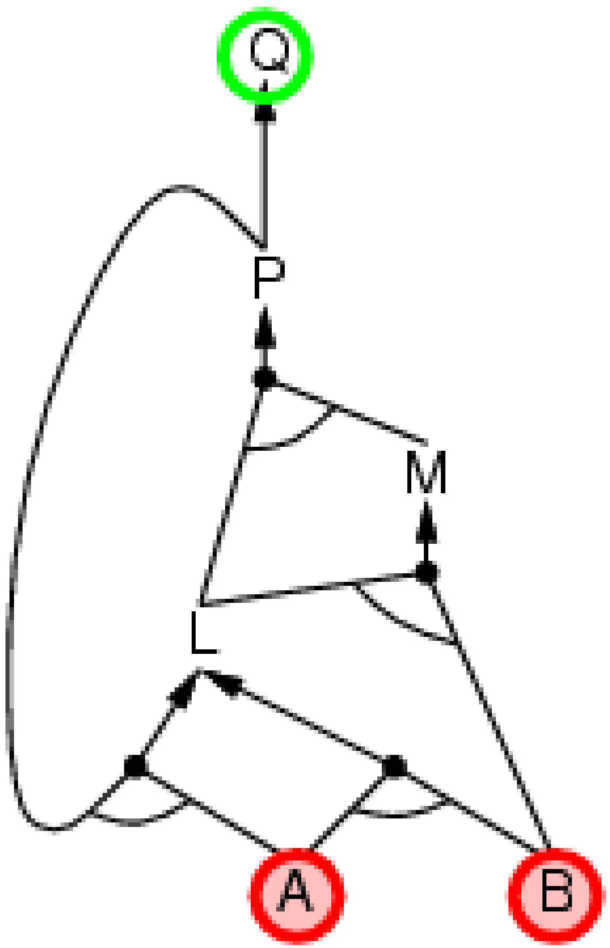
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

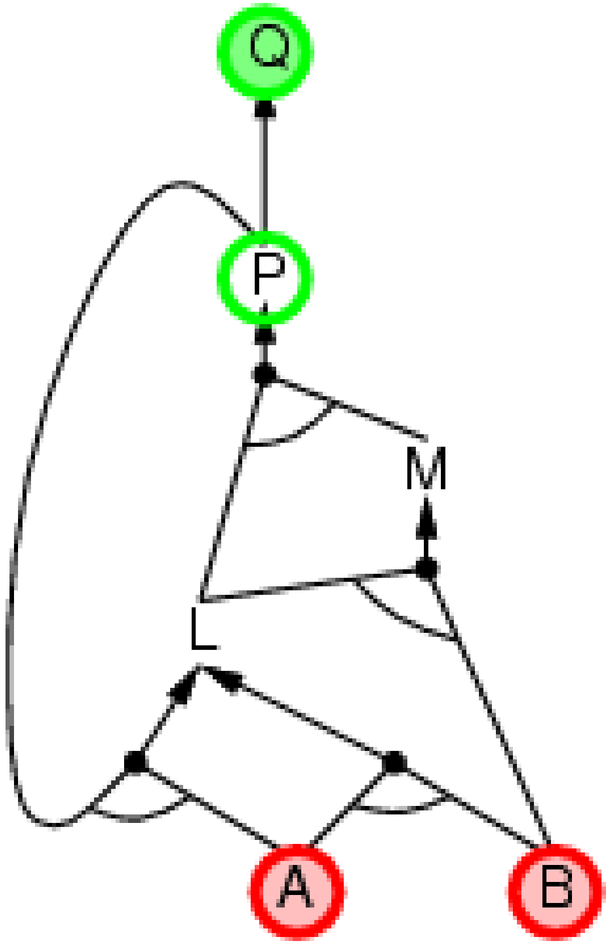
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

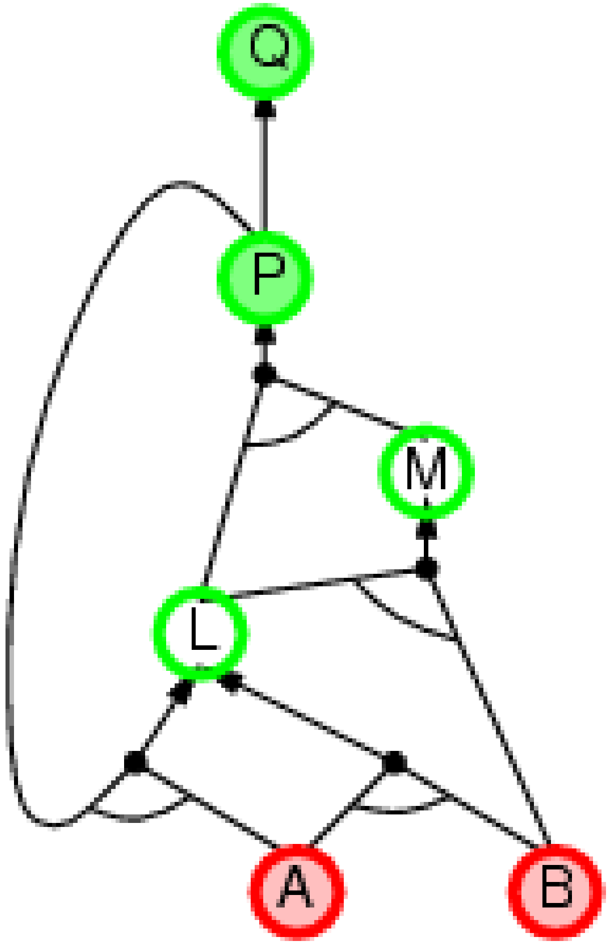
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

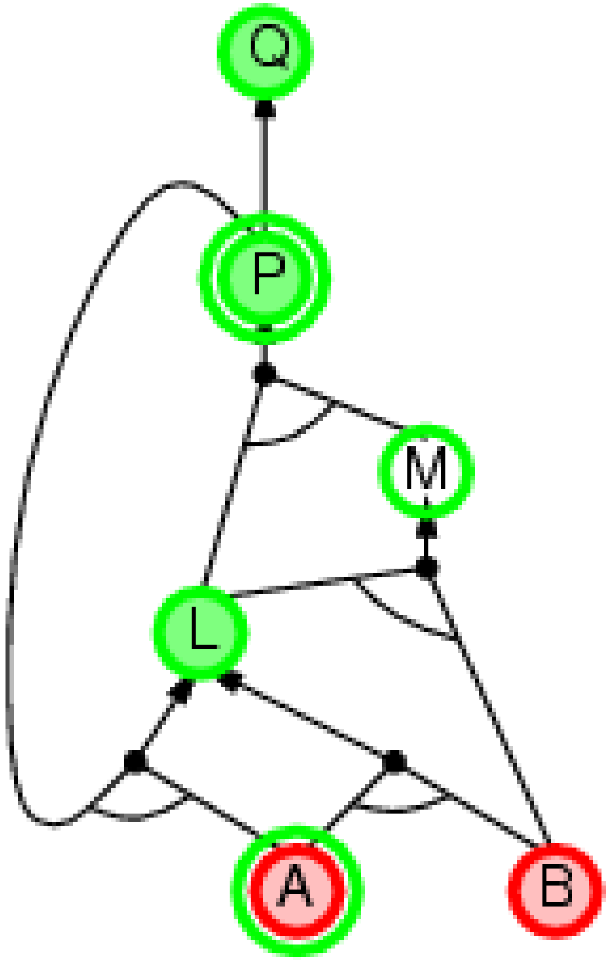
A

B



Backward chaining 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



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

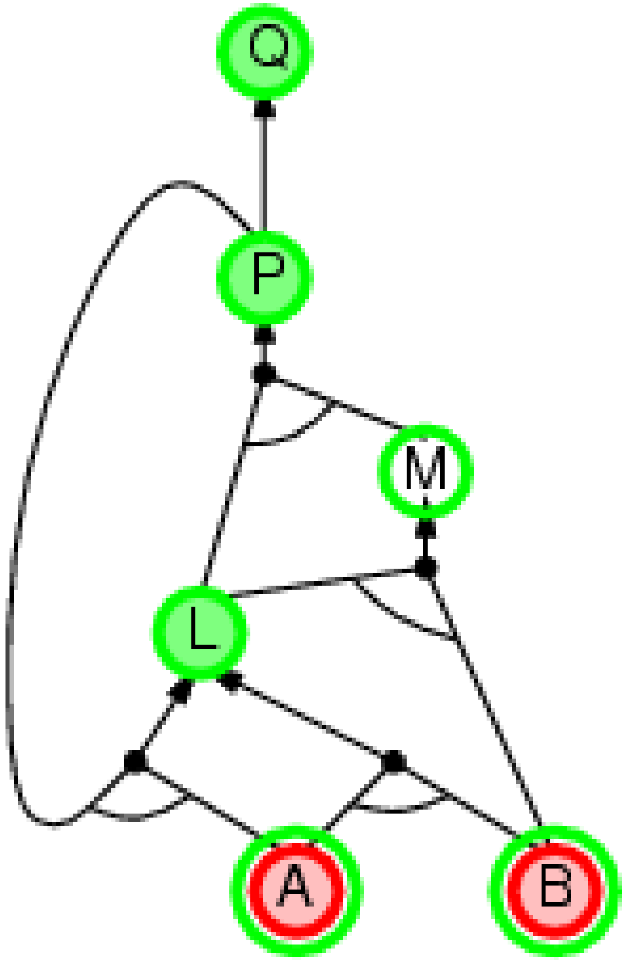
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

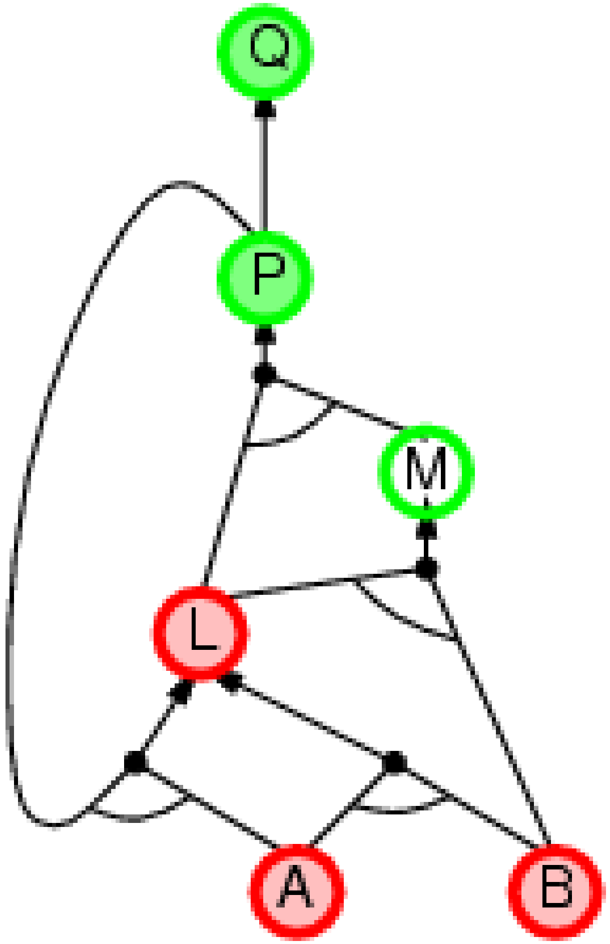
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

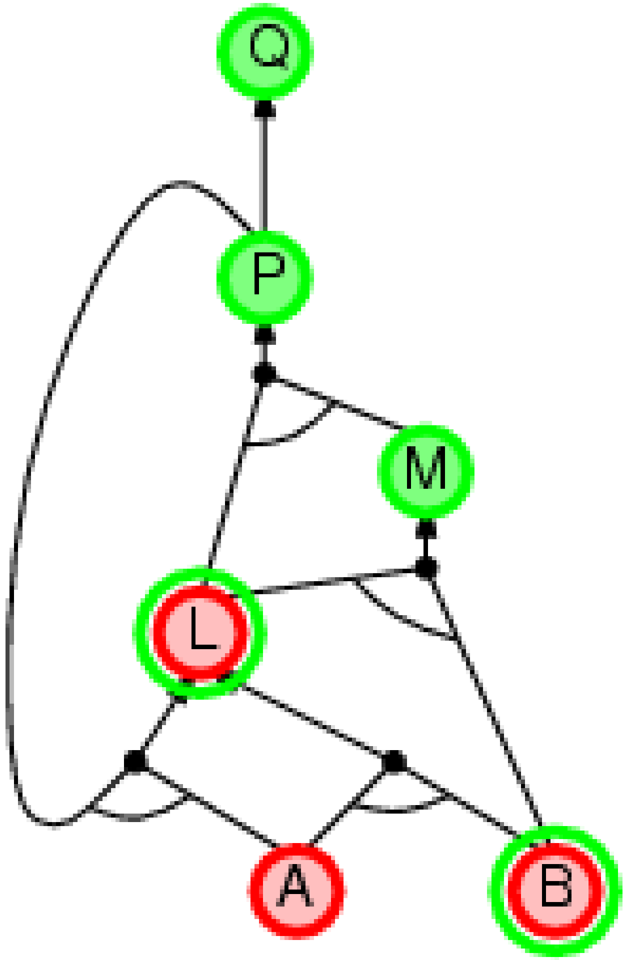
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

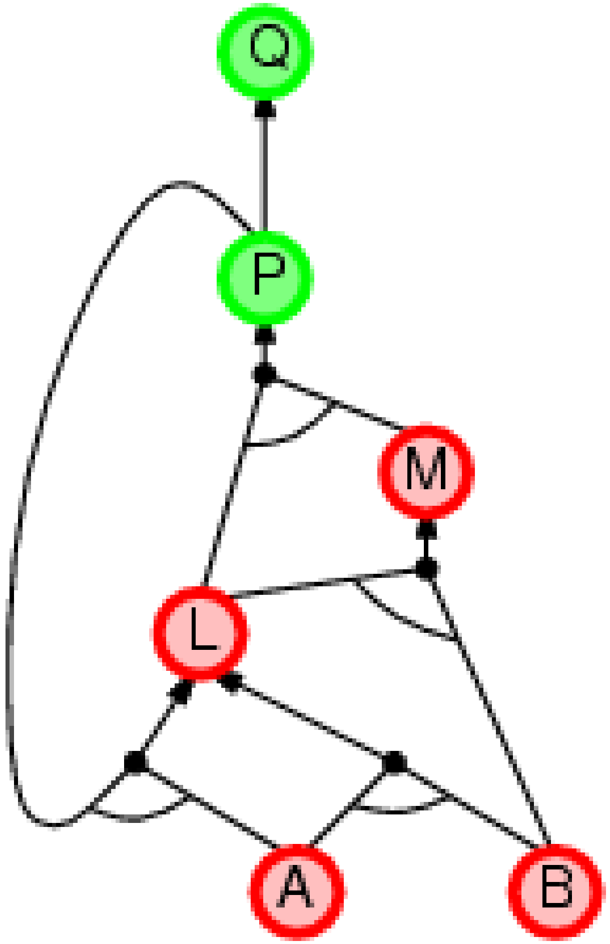
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

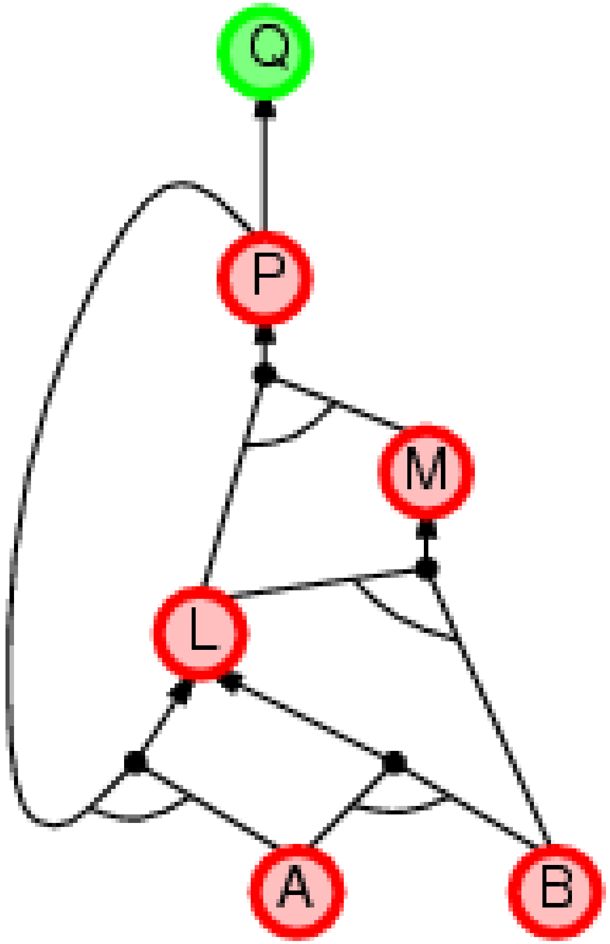
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

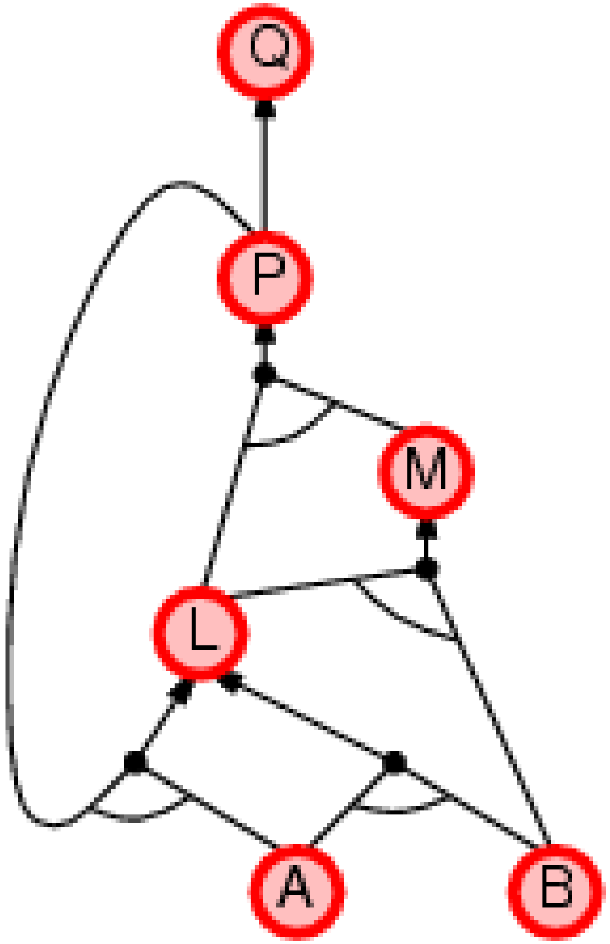
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



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**

Efficient propositional inference

Two families of efficient algorithms for propositional inference:

Complete backtracking search algorithms

- DPLL algorithm (Davis, Putnam, Logemann, Loveland)

Incomplete local search algorithms

- `WalkSAT` algorithm

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

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 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.

3. Unit clause heuristic

   Unit clause: only one literal in the clause

   The only literal in a unit clause must be true.

The DPLL algorithm

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, [ ])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [P = *value*|*model*])
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [P = *value*|*model*])
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, [P = *true*|*model*]) **or**
      DPLL(*clauses*, *rest*, [P = *false*|*model*])

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

The `WalkSAT` algorithm

function WALKSAT(*clauses*, *p*, *max-flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move
    max-flips, number of flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for $i = 1$ **to** *max-flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability *p* flip the value in *model* of a randomly selected symbol
      from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

Hard satisfiability problems

- Consider random 3-CNF sentences. e.g.,

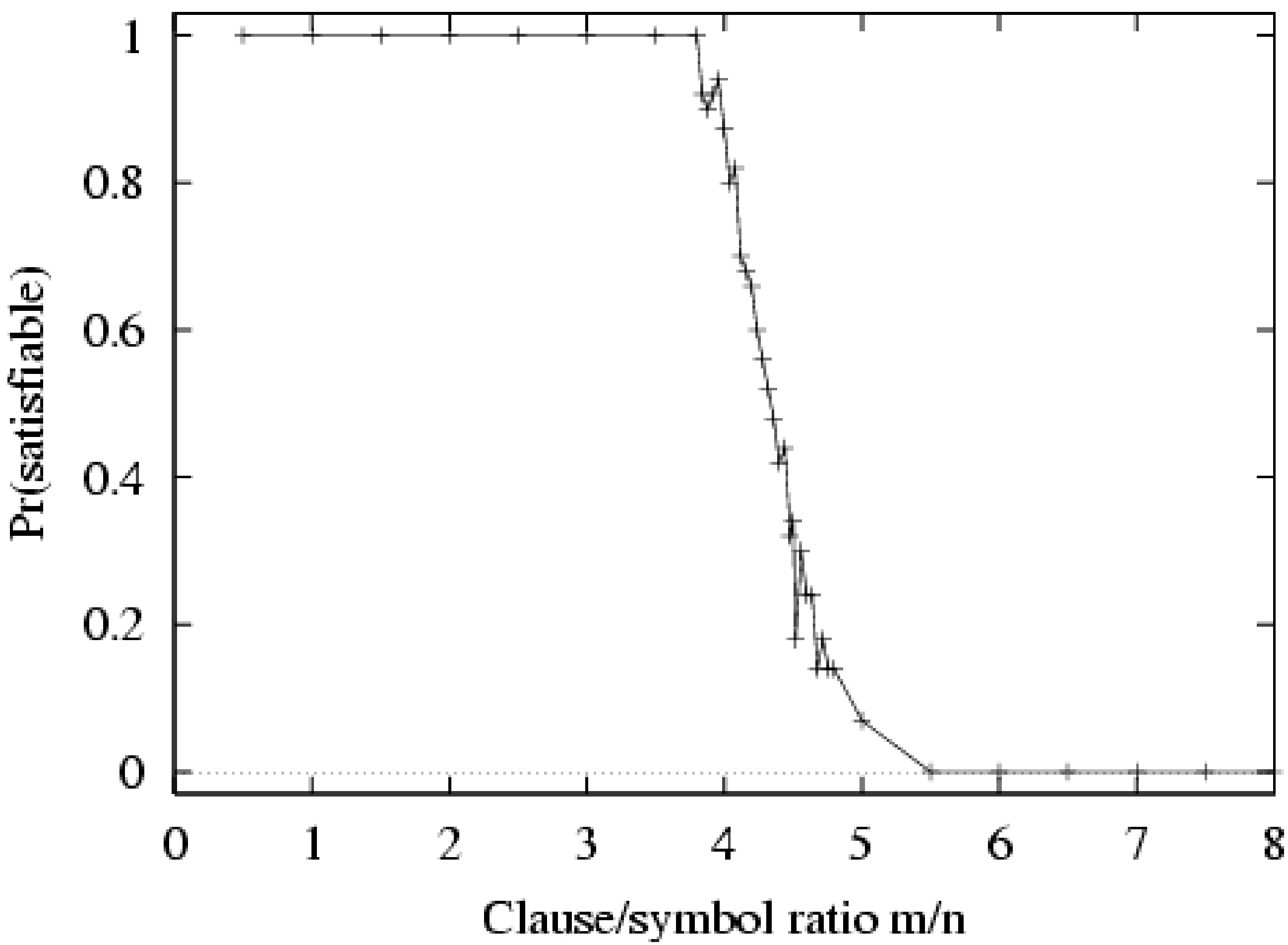
  $(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$

  m = number of clauses

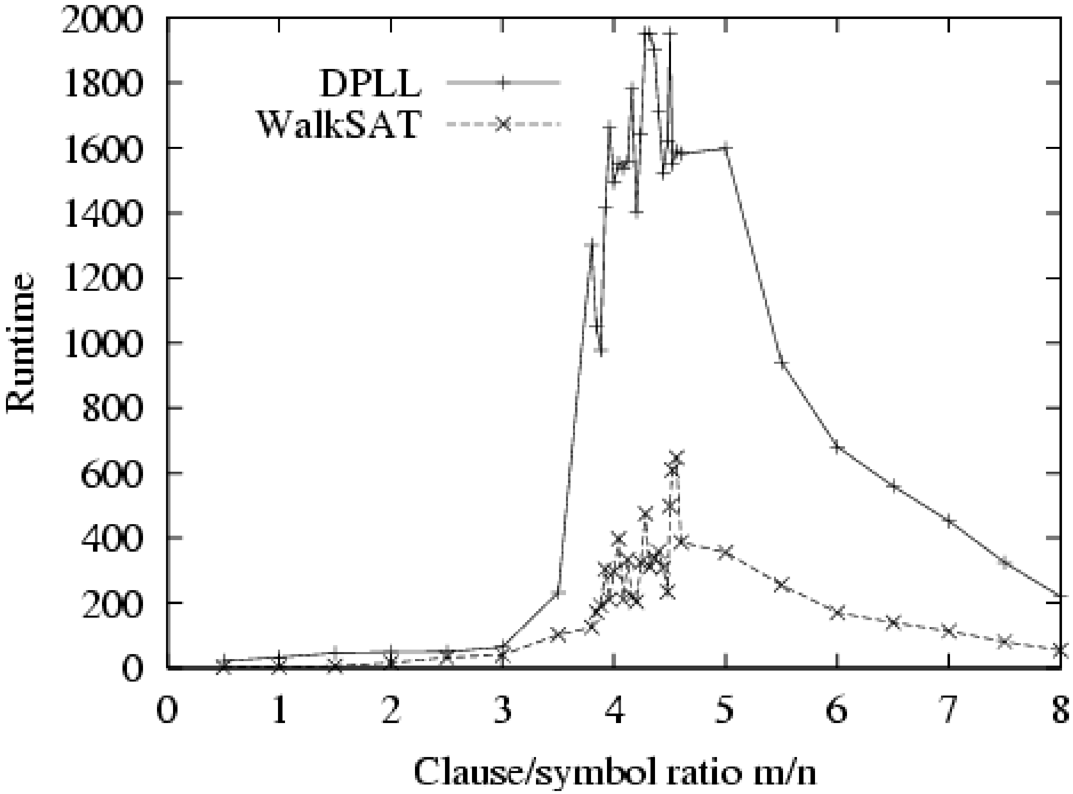
  n = number of symbols

  – Hard problems seem to cluster near *m/n* = 4.3 (critical point) (called a *phase transition*)

Hard satisfiability problems



Hard satisfiability problems



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

Inference-based agents in wumpus

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$
$\neg W_{1,1}$
$B_{x,y} \Leftrightarrow (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y})$
$W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,4}$
$\neg W_{1,1} \lor \neg W_{1,2}$
$\neg W_{1,1} \lor \neg W_{1,3}$
…

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences

```
function PL-WUMPUS-AGENT(percept) returns an action
   inputs: percept, a list, [stench,breeze,glitter]
   static: KB, initially containing the "physics" of the wumpus world
           x, y, orientation, the agent's position (init. [1,1]) and orient. (init. right)
           visited, an array indicating which squares have been visited, initially false
           action, the agent's most recent action, initially null
           plan, an action sequence, initially empty

   update x,y,orientation, visited based on action
   if stench then TELL(KB, S_x,y) else TELL(KB, ¬ S_x,y)
   if breeze then TELL(KB, B_x,y) else TELL(KB, ¬ B_x,y)
   if glitter then action ← grab
   else if plan is nonempty then action ← POP(plan)
   else if for some fringe square [i,j], ASK(KB, (¬ P_i,j ∧ ¬ W_i,j)) is true or
           for some fringe square [i,j], ASK(KB, (P_i,j ∨ W_i,j)) is false then do
       plan ← A*-GRAPH-SEARCH(ROUTE-PB([x,y], orientation, [i,j],visited))
       action ← POP(plan)
   else action ← a randomly chosen move
   return action
```

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
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Forward, backward chaining are linear-time, complete for Horn clauses
Resolution is complete for propositional logic

Propositional logic lacks expressive power