



# 人工智能



罗平 [luop@ict.ac.cn](mailto:luop@ict.ac.cn)

# Knowledge 2



# RESOLUTION 归结原理

有限

# Resolution (消解、归结)

## ■ Conjunctive Normal Form (CNF—universal)

conjunction of **disjunctions of literals** (clauses)

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

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

$\ell_1 \vee \dots \vee \ell_k,$

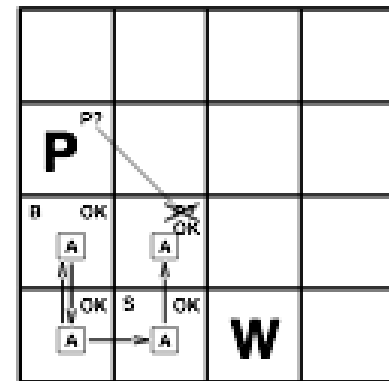
$m_1 \vee \dots \vee m_n$

$\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n$

where  $\ell_i$  and  $m_j$  are complementary literals. E.g.,

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

- 反子



## ■ Resolution is sound and complete for propositional logic

# 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:

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


$\{ \neg B_{1,1} \vee P_{1,2} \vee P_{2,1}, \neg P_{1,2} \vee B_{1,1}, \neg P_{2,1} \vee B_{1,1} \}$  KB'

多项式时间复杂度

# Resolution algorithm

- Proof by contradiction, i.e., show  $KB \wedge \neg\alpha$  unsatisfiable

```
function PL-RESOLUTION( $KB, \alpha$ ) returns true or false
  inputs:  $KB$ , the knowledge base, a sentence in propositional logic
          $\alpha$ , the query, a sentence in propositional logic
   $clauses \leftarrow$  the set of clauses in the CNF representation of  $KB \wedge \neg\alpha$ 
   $new \leftarrow \{ \}$ 
  loop do
    for each  $C_i, C_j$  in  $clauses$  do
       $resolvents \leftarrow$  PL-RESOLVE( $C_i, C_j$ )
      if  $resolvents$  contains the empty clause then return true
       $new \leftarrow new \cup resolvents$ 
  if  $new \subseteq clauses$  then return false
   $clauses \leftarrow clauses \cup new$ 
```



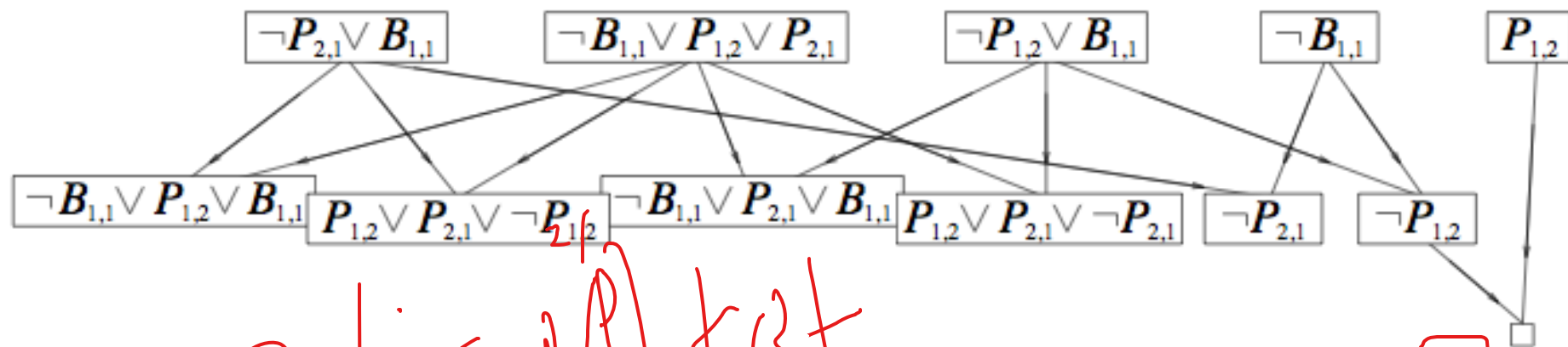
# Resolution example

证明:  $KB \vdash \alpha$

取反结论出  
矛盾

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

$$\alpha = \neg P_{1,2}$$



证明 KB

矛盾

# Resolution is sound

证明: Resolution 规则是可靠的。即证明:

■ 证明: **Resolution** 规则是可靠的。即证明:

■  $(l_1 \vee \dots \vee l_k) \wedge (m_1 \vee \dots \vee m_n)$

$\models (l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)$

$l_i$  与  $m_j$  互补

为真

Resolution 规则:

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

where  $l_i$  and  $m_j$  are complementary literals.

2个命题 / 1个命题 矛盾



# Resolution is complete

- Soundness is not surprising since inference rules are sound (check the truth table)
- Resolution is also complete.
  - Resolution closure  $RC(S)$  of a set of clauses  $S$ : the set of all clauses derivable by resolution
  - Final value of *clauses* in **PL\_RESOLUTION** is  $RC(S)$
  - $RC(S)$  is finite, and hence **PL\_RESOLUTION** always terminates.

## Ground Resolution Theorem

$S$  is unsatisfiable  $\Rightarrow RC(S)$  contains the empty clause.

$$S = \{KB, \neg\alpha\}$$

$$KB \models \alpha$$

# Ground resolution theorem

$RC(S)$  does not contain the empty set  $\Rightarrow S$  is satisfiable.

- 证明：针对  $S$  中的原子命题  $R_1, R_2, \dots, R_l$ ，我们构造如下的 model:
- 首先，因为  $RC(S)$  中不包含空集，即  $RC(S)$  中不包含永假的子句。

举例：第7页的例子

从  $i = 1$  到  $l$ ，顺序的指派  $R_1, R_2, \dots, R_l$  的真值：

如果  $RC(S)$  中包含一个子句，此子句包含  $\neg R_i$ ，且此子句的其它文字已经被指派为 **False**（在之前的步骤中进行的），则把  $R_i$  指派为 **False**；

否则，把  $R_i$  指派为 **True**

- 我们用反证法证明：这个 model 使得  $RC(S)$  中的子句都为真。假设，在此过程的第  $i$  步，我们这样来指派  $R_i$  使得某个子句 **C** 为 **False**（假设这是首次出现 **False** 的子句）；此时，子句 **C** 只能是如下两种形式之一：

$False \vee False \cdots \vee False \vee R_i$  或者  $False \vee False \cdots \vee False \vee \neg R_i$

- 显然，如果  $RC(S)$  中只包含以上两个子句之一，子句 **C** 是不会在此 model 中为 **False** 的。因此， $RC(S)$  此时应该同时包含了以上两个子句。
- 以上两个子句显然是满足归结条件的，也就是说，它归结后的子句也应该在  $RC(S)$  中；同时，该子句已经被指派为 **False** 了；这与我们之前的假设 **矛盾**。

# Process of Resolution: Search

Path-based Search: goal, actions

Requirement: optimal solution in terms of the number of resolution steps

Homework: design a heuristic for A\* search

Requirements: formally define what actions are (single clause or **a set of clauses (preferred)**)

手工A\* 算法

# Horn and Definite Clauses

## Alfred Horn

---

From Wikipedia, the free encyclopedia

**Alfred Horn** (February 17, 1918 – April 16, 2001) was an [American mathematician](#) notable for his work in [lattice theory](#) and [universal algebra](#). His 1951 paper "On sentences which are true of direct unions of algebras" described [Horn clauses](#) and [Horn sentences](#), which later would form the foundation of [logic programming](#).

[https://en.wikipedia.org/wiki/Alfred\\_Horn](https://en.wikipedia.org/wiki/Alfred_Horn)

# Horn and Definite Clauses

什么是正文字和负文字？

- The completeness of resolution is good.
- For many real-world applications, if we add some restrictions, more efficient inference can be achieved.
- **Definite clause**: a disjunction of literals where **exactly one** is positive
- **Horn clause**: a disjunction of literals where **at most one** is positive
- Horn clauses are **closed** under resolution:

Resolving two Horn clauses yields a Horn clause.

- Another way to view Horn clauses:

TRUE  $\Rightarrow$  symbol

(Conjunction of symbols)  $\Rightarrow$  symbol

- Deciding entailment with Horn clauses can be done in **linear** time!

Forward and backward chaining

缩小propositional logic的表达范围，以换取更好的inference的时间效率

# Forward and backward chaining

- **Horn Form** (restricted)

KB = conjunction of definite clauses

definite 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

(肯定式推理)

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

归结的一种形式

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

# Forward chaining (前向推理)

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

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

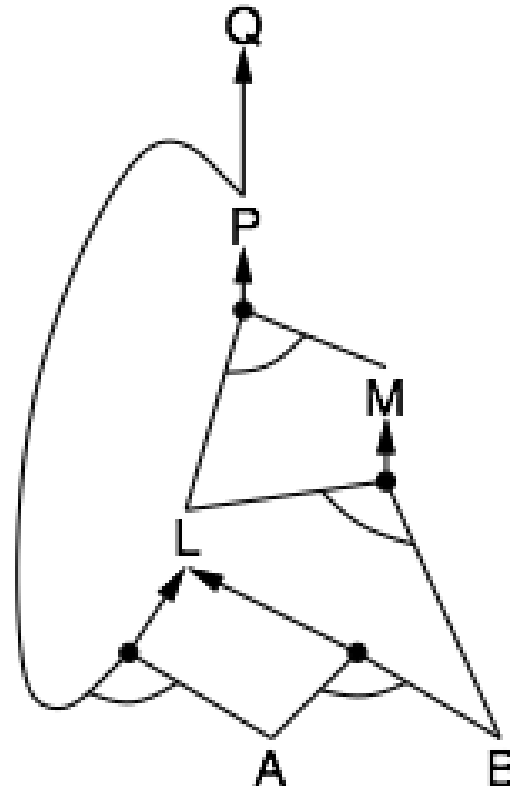
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

*A*

*B*



# Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional Horn clauses
         q, the query, a proposition symbol
  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 in KB

  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
```



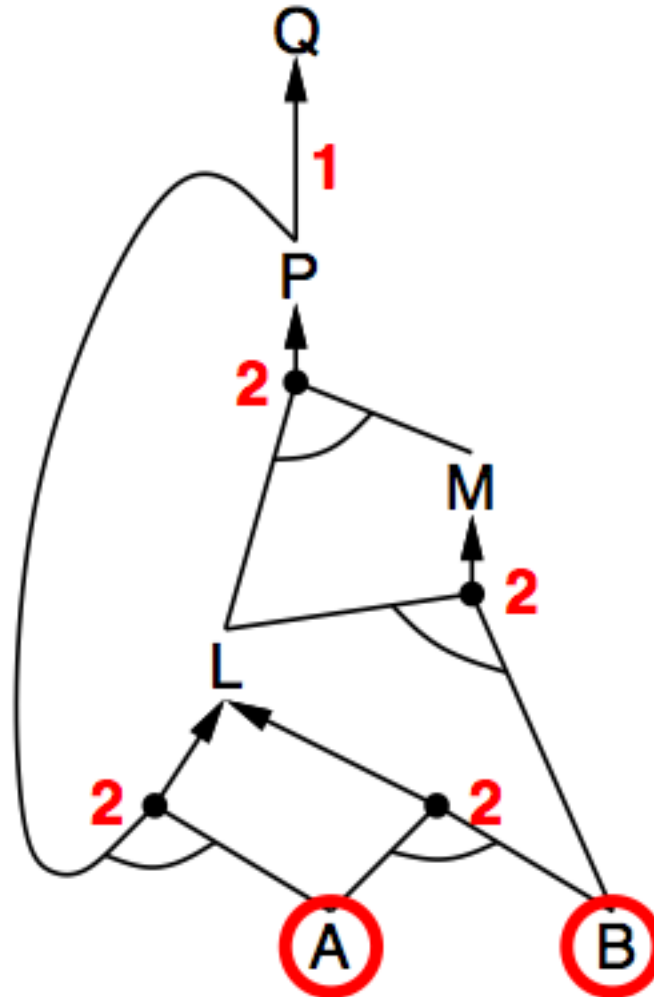
# Forward chaining example

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

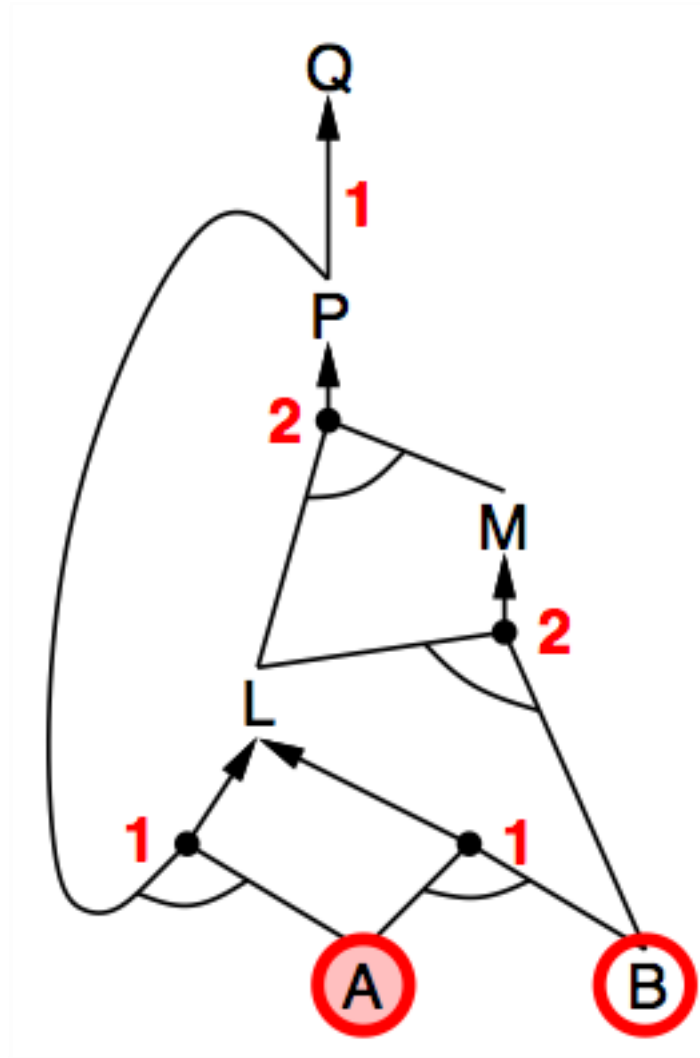
KB	Step 1	Step 2	Step 3
$P \Rightarrow Q$ $L \wedge M \Rightarrow P$ $B \wedge L \Rightarrow M$ $A \wedge P \Rightarrow L$ $A \wedge B \Rightarrow L$ $A$ $B$	$P \Rightarrow Q$ 1 $L \wedge M \Rightarrow P$ 2 $B \wedge L \Rightarrow M$ 2 $A \wedge P \Rightarrow L$ 2 $A \wedge B \Rightarrow L$ 2 <i>agenda</i> : [A, B]	$P \Rightarrow Q$ 1 $L \wedge M \Rightarrow P$ 2 $B \wedge L \Rightarrow M$ 2 $A \wedge P \Rightarrow L$ 1 $A \wedge B \Rightarrow L$ 1 <i>agenda</i> : [B]	$P \Rightarrow Q$ 1 $L \wedge M \Rightarrow P$ 2 $B \wedge L \Rightarrow M$ 1 $A \wedge P \Rightarrow L$ 1 $A \wedge B \Rightarrow L$ 0 <i>agenda</i> : [L]
Step 4	Step 5	Step 6	
$P \Rightarrow Q$ 1 $L \wedge M \Rightarrow P$ 1 $B \wedge L \Rightarrow M$ 0 $A \wedge P \Rightarrow L$ 1 $A \wedge B \Rightarrow L$ 0 <i>agenda</i> : [M]	$P \Rightarrow Q$ 1 $L \wedge M \Rightarrow P$ 0 $B \wedge L \Rightarrow M$ 0 $A \wedge P \Rightarrow L$ 1 $A \wedge B \Rightarrow L$ 0 <i>agenda</i> : [P]	$P \Rightarrow Q$ 0 $L \wedge M \Rightarrow P$ 0 $B \wedge L \Rightarrow M$ 0 $A \wedge P \Rightarrow L$ 0 $A \wedge B \Rightarrow L$ 0 <i>agenda</i> : [Q, L]	

Linear to the number of what?

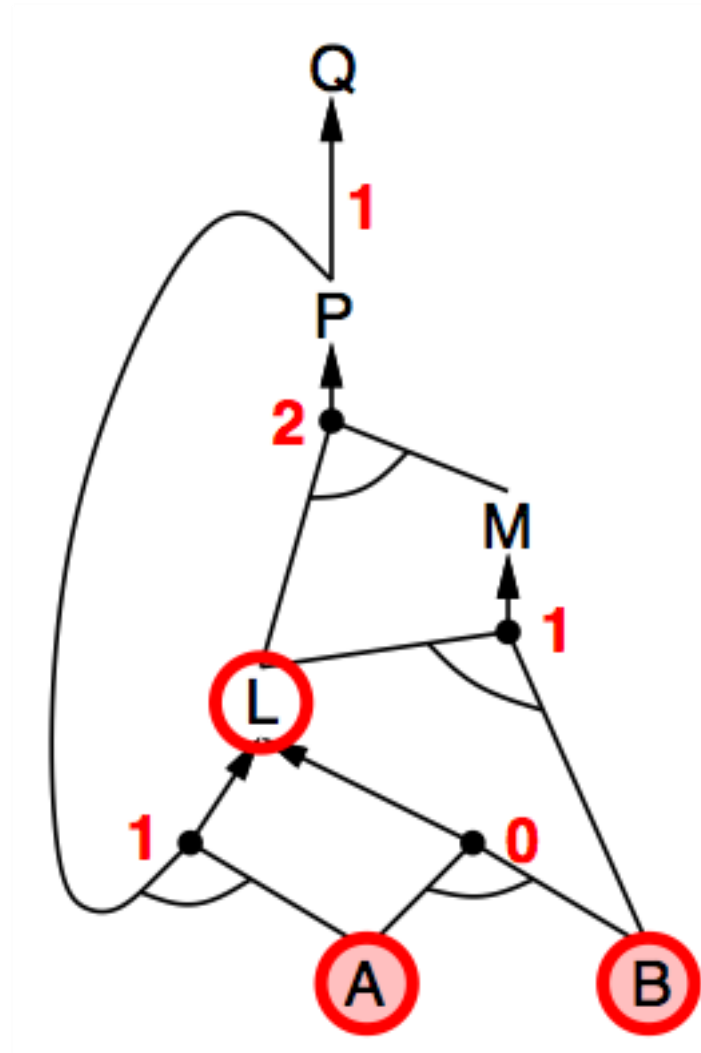
# Forward chaining example



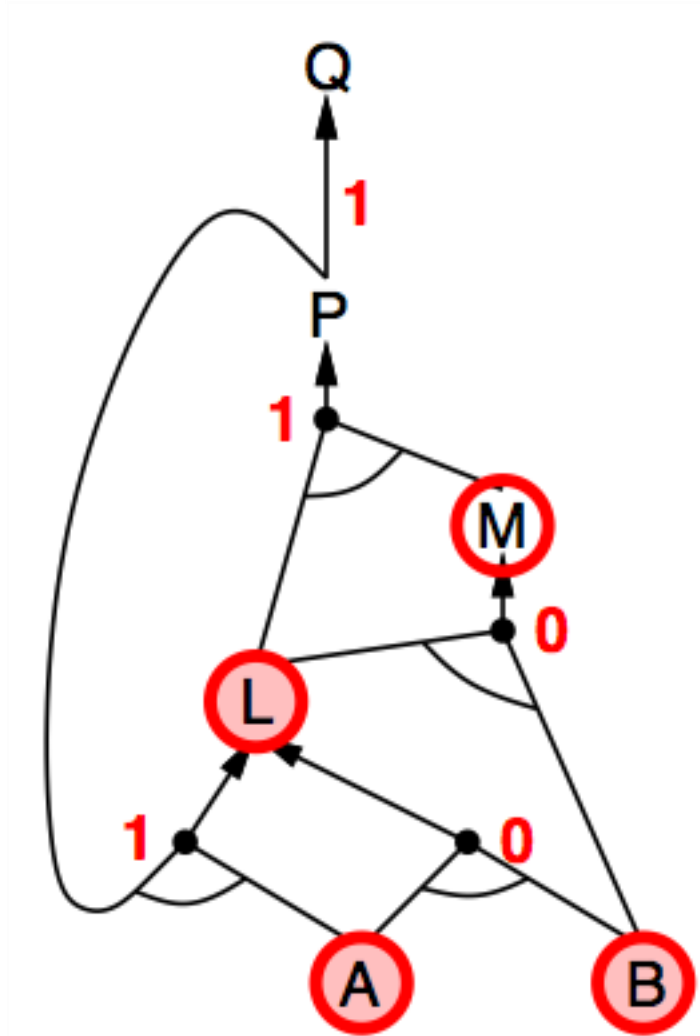
# Forward chaining example



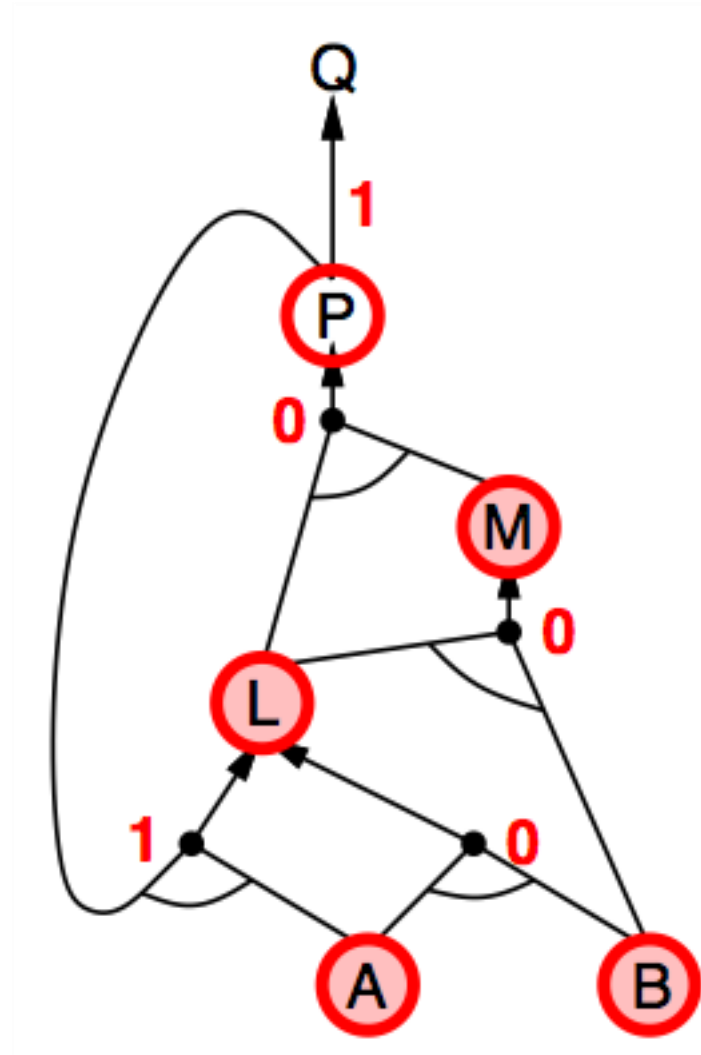
# Forward chaining example



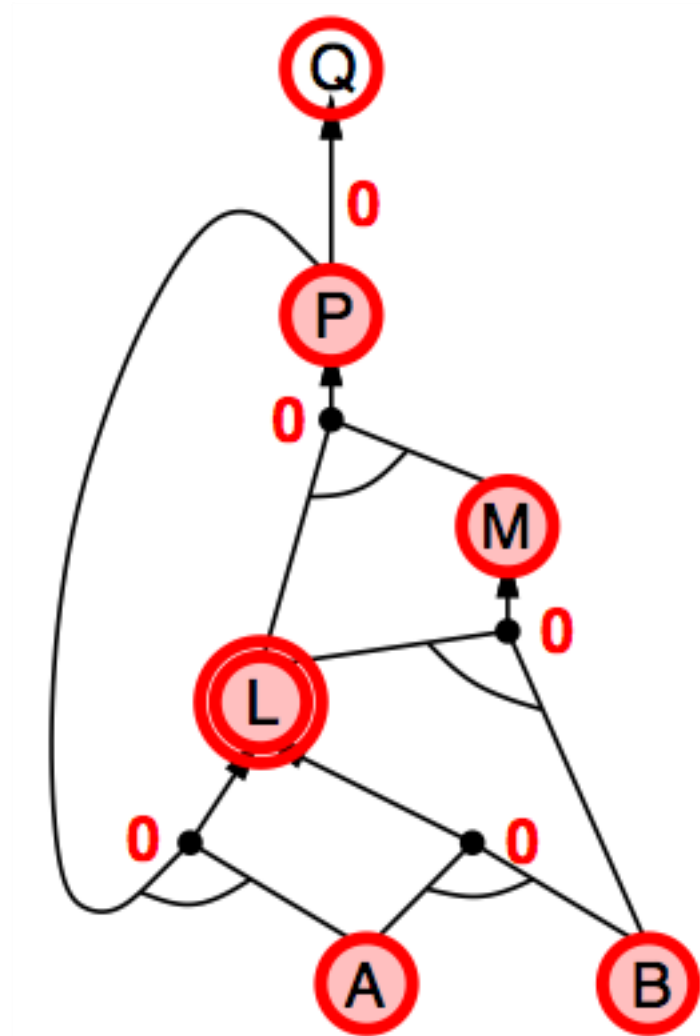
# Forward chaining example



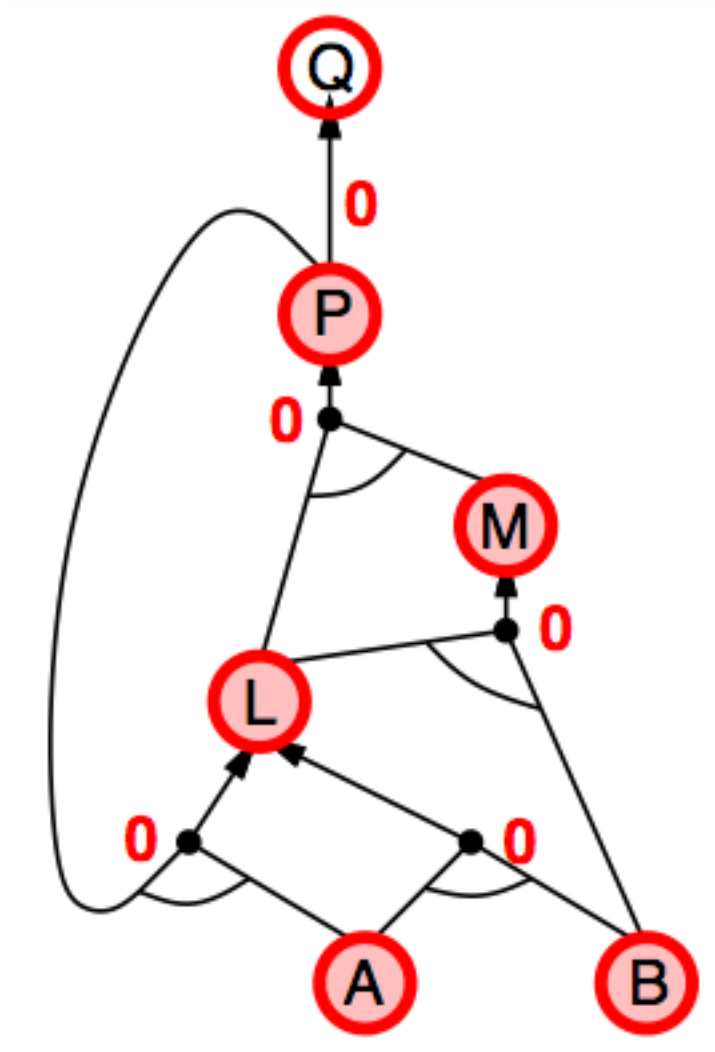
# Forward chaining example



# Forward chaining example

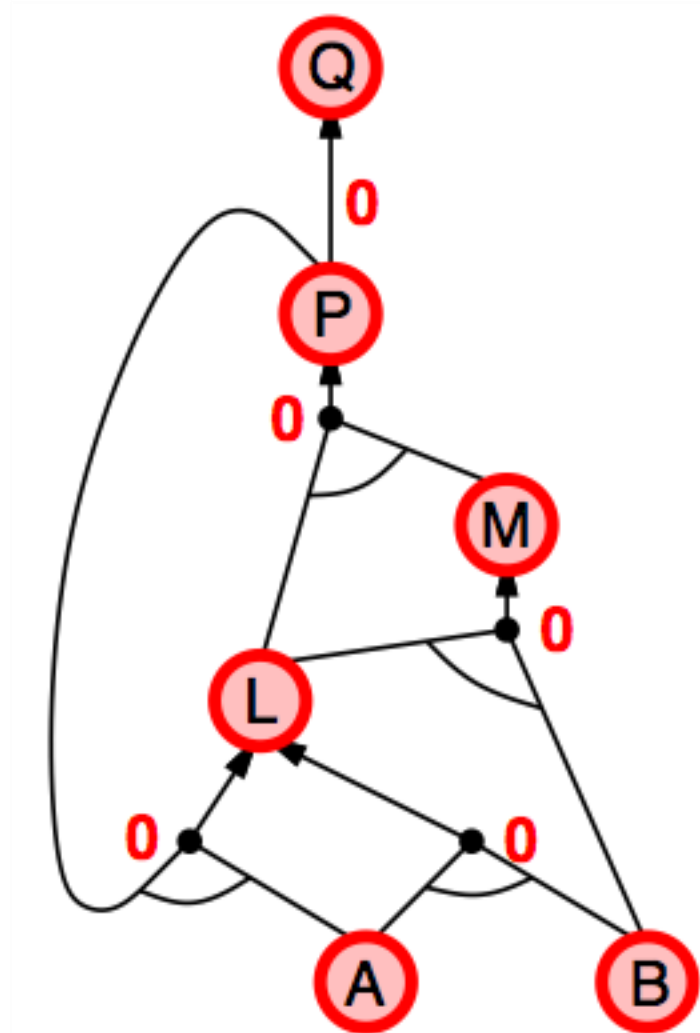


# Forward chaining example





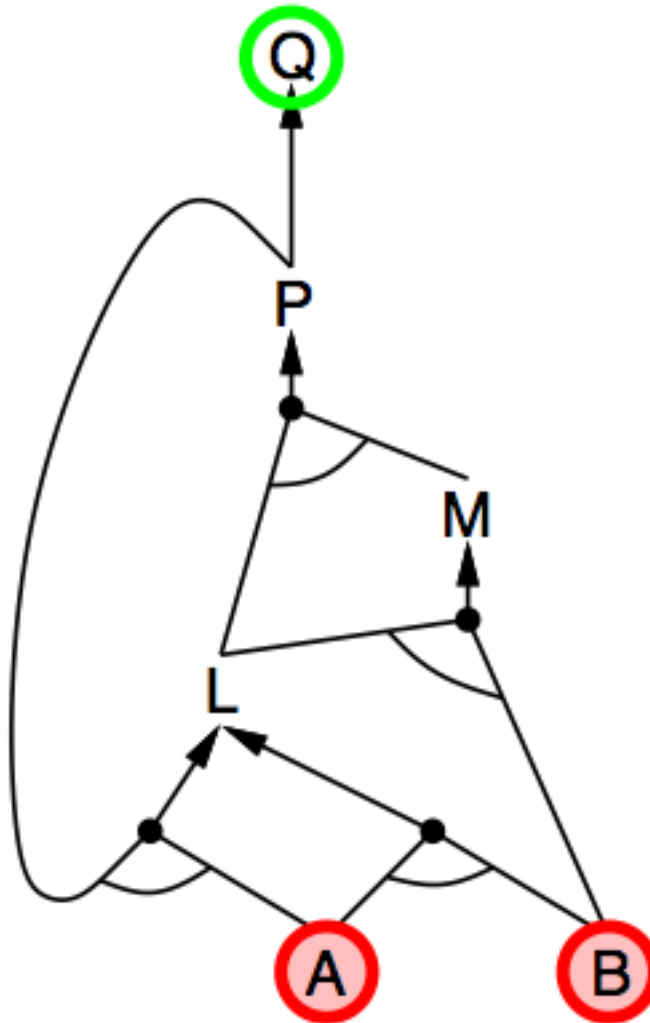
# Forward chaining example



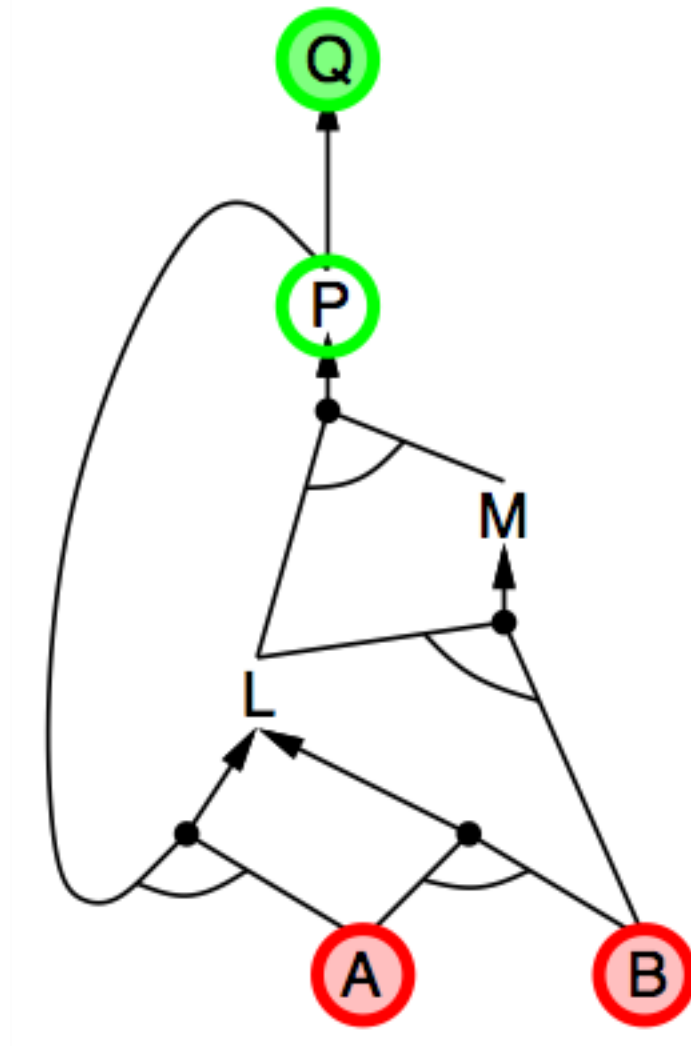
# Backward chaining (后向推理)

- 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

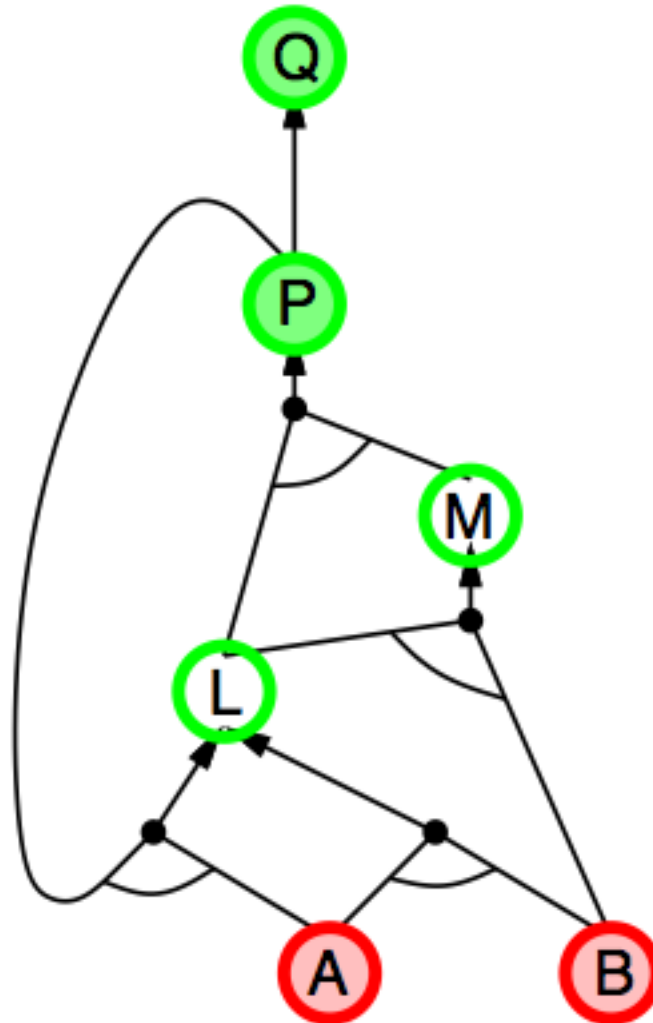
# Backward chaining example



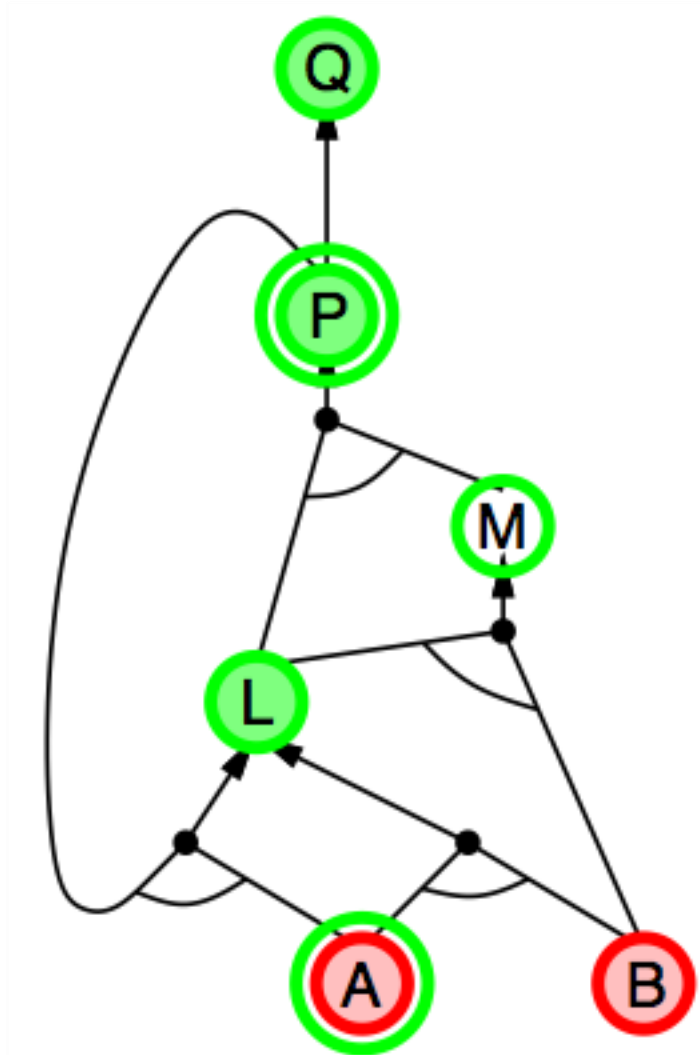
# Backward chaining example



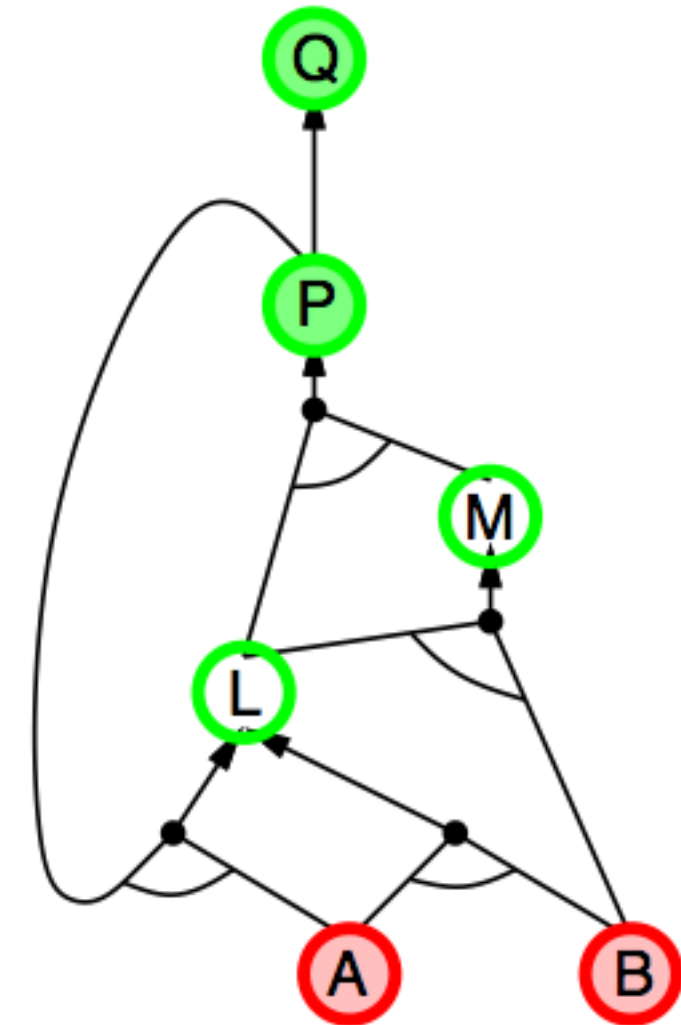
# Backward chaining example



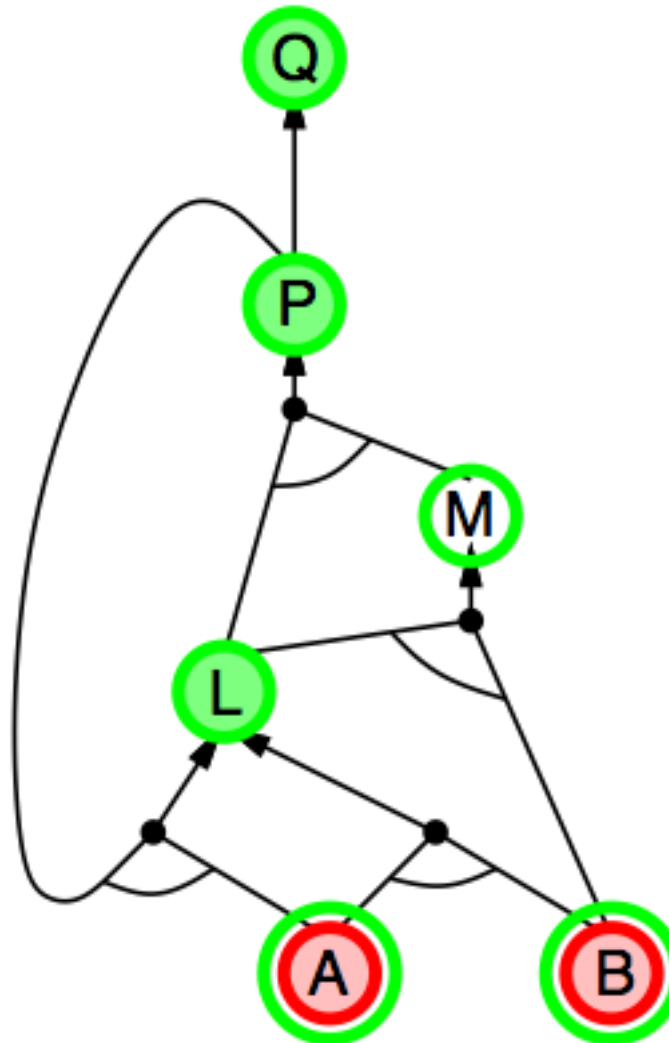
# Backward chaining example



# Backward chaining example

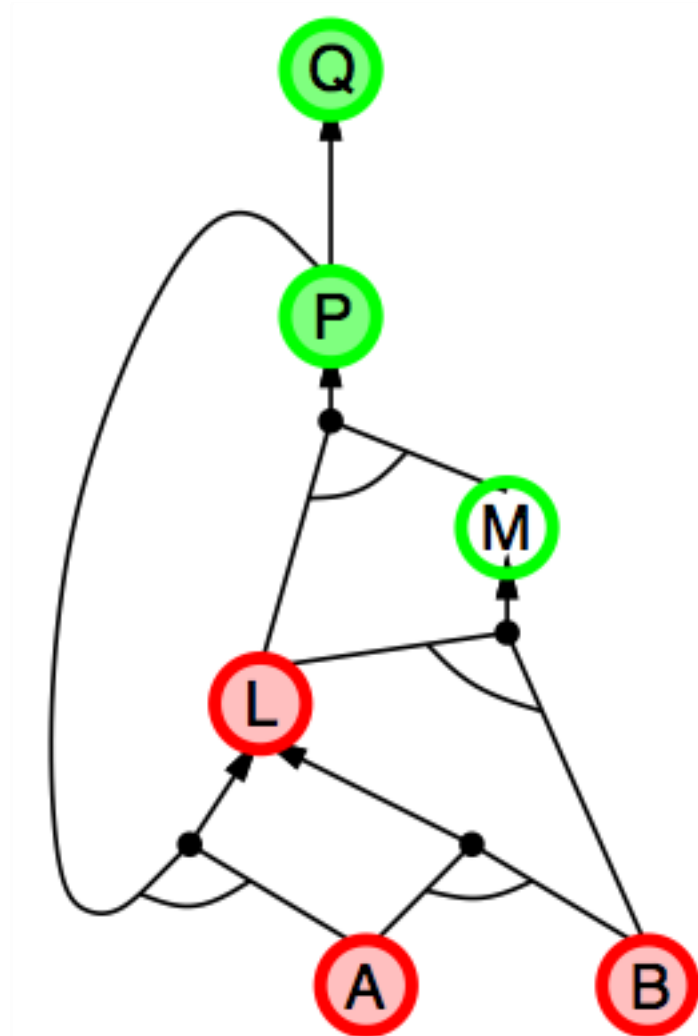


# Backward chaining example

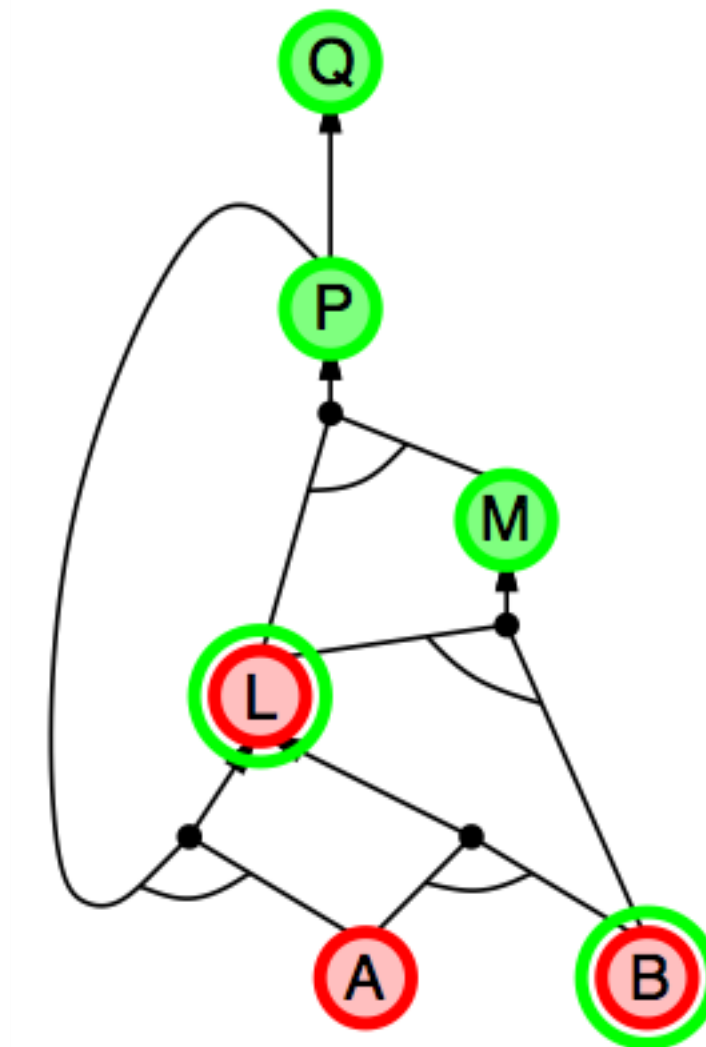




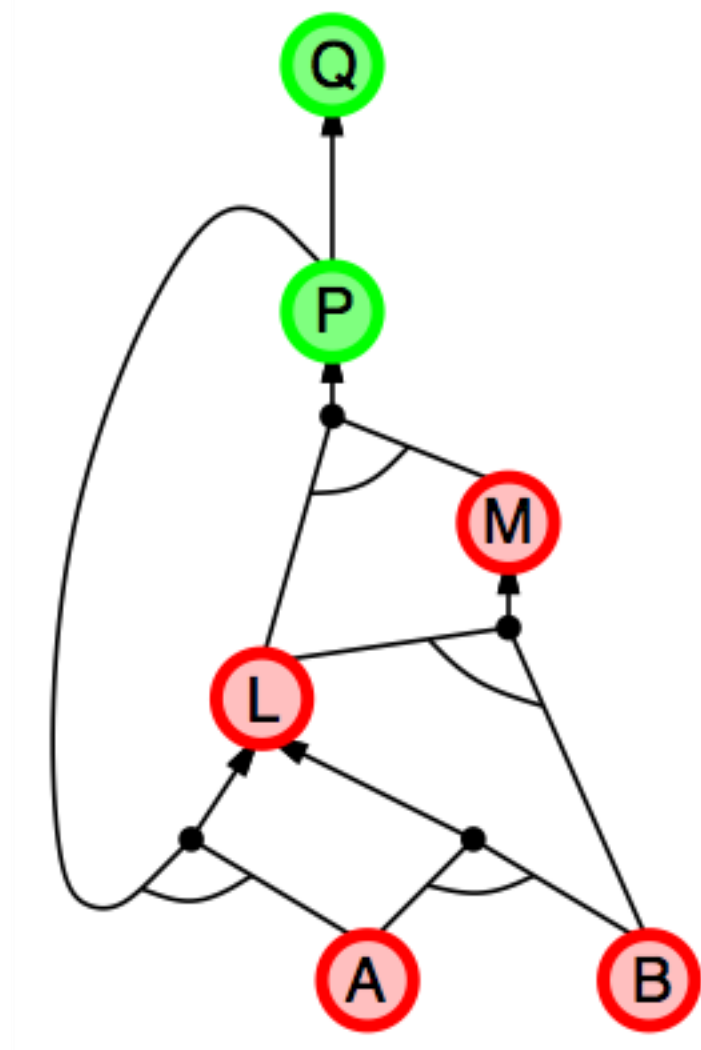
# Backward chaining example



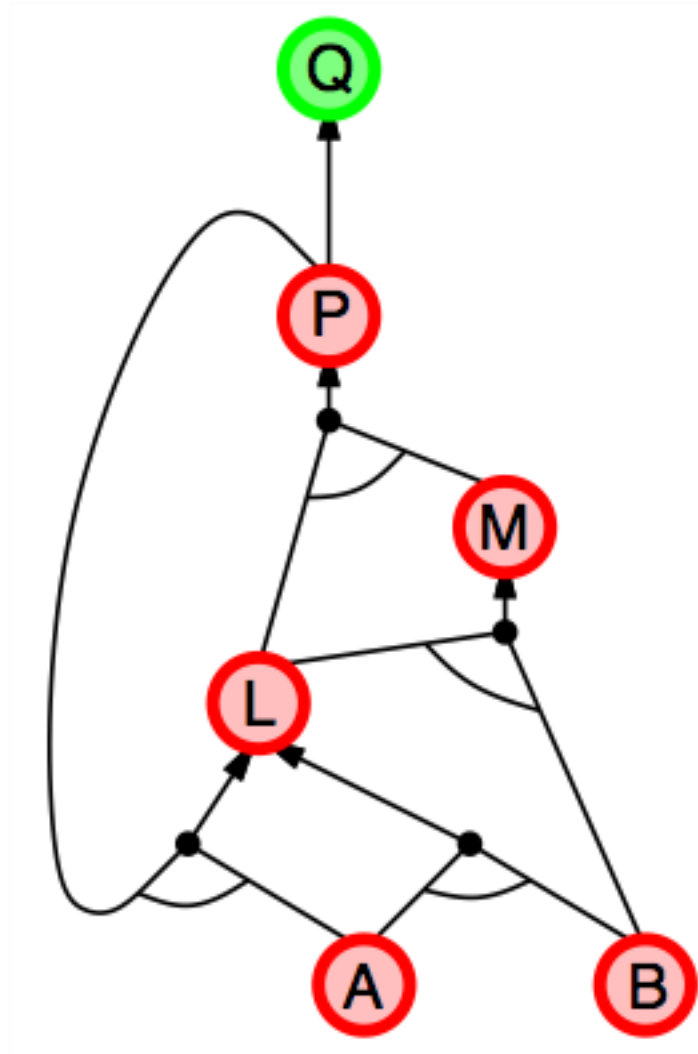
# Backward chaining example



# Backward chaining example



# Backward chaining example



# Comparison

- FC is **data-driven**, cf. 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

# Proof of soundness

- 证明: **Modus Ponens**规则是可靠的。即证明:

$$\alpha_1 \wedge \cdots \wedge \alpha_n \wedge (\alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta) \models \beta$$

Modus Ponens规则:

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

# Proof of completeness

- 若  $KB \models \alpha$ ,  $KB \vdash \alpha$ 。此时,  $KB$  中仅包含 **definite** 子句,  $\vdash$  仅使用 **Modus Ponens** 规则, 且  $\alpha$  是一个正文字
- **证明:**  $RC(KB)$  is the set of all clauses derived by Modus Ponens and all the original clauses inside  $KB$
- 1) 构造如下的真值指派  $m$ : 对于任意的 symbol  $a$ ,  
     $a$  指派为 True 当且仅当  $a \in RC(KB)$
- 2) 接下来, 证明: 在  $m$  下,  $KB$  为真。  
    反证: 若此时  $KB$  为 False, 那么:  
    必存在一个 definite 子句, 在  $m$  下为 False。  
    i) 若该子句为  $a_1 \wedge \dots \wedge a_k \Rightarrow b$   
    也就是说, 在  $m$  中,  $a_1, \dots, a_k$  均为 True, 且  $b$  为 False。  
    根据 1) 中的定义,  $a_i \in RC(KB)$  ( $i=1, \dots, k$ )  
    又根据 Modus Ponens 规则,  $b \in RC(KB)$   
    根据 1) 中的定义, 在  $m$  中,  $b$  为 True。推出矛盾。  
    ii) 若该子句为  $b$ , 在  $m$  下为  $b$  为 False, 则  $b \notin RC(KB)$ , 矛盾
- 3) 若  $KB \models \alpha$ , 根据蕴含的定义: 在  $m$  中,  $\alpha$  为真;  
    则根据 1) 中的定义,  $\alpha \in RC(KB)$   
    也就是说:  $KB \vdash \alpha$

# Pros and Cons of Propositional Logic

## ■ Pro

- Propositional logic is **declarative**: pieces of syntax correspond to facts.
- Propositional logic allows partial/disjunctive/negated information (unlike most data structures and databases)
- Propositional logic is **compositional**:  
Meaning of  $B_{1,1} \wedge P_{1,2}$  is derived from meaning of  $B_{1,1}$  and of  $P_{1,2}$
- Meaning in propositional logic is **context-independent** (unlike natural language, where meaning depends on context)

## ■ Con

- Propositional has very limited expressive power
  - Cannot say “pits cause breezes in adjacent squares” except by writing one sentence for each square



# 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

**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**

# Homework

**7.17** A propositional 2-CNF expression is a conjunction of clauses, each containing *exactly* 2 literals, e.g.,

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

- a. Prove using resolution that the above sentence entails  $G$ .
- b. Two clauses are *semantically distinct* if they are not logically equivalent. How many semantically distinct 2-CNF clauses can be constructed from  $n$  proposition symbols?
- c. Using your answer to (b), prove that propositional resolution always terminates in time polynomial in  $n$  given a 2-CNF sentence containing no more than  $n$  distinct symbols.
- d. Explain why your argument in (c) does not apply to 3-CNF.