人工智能

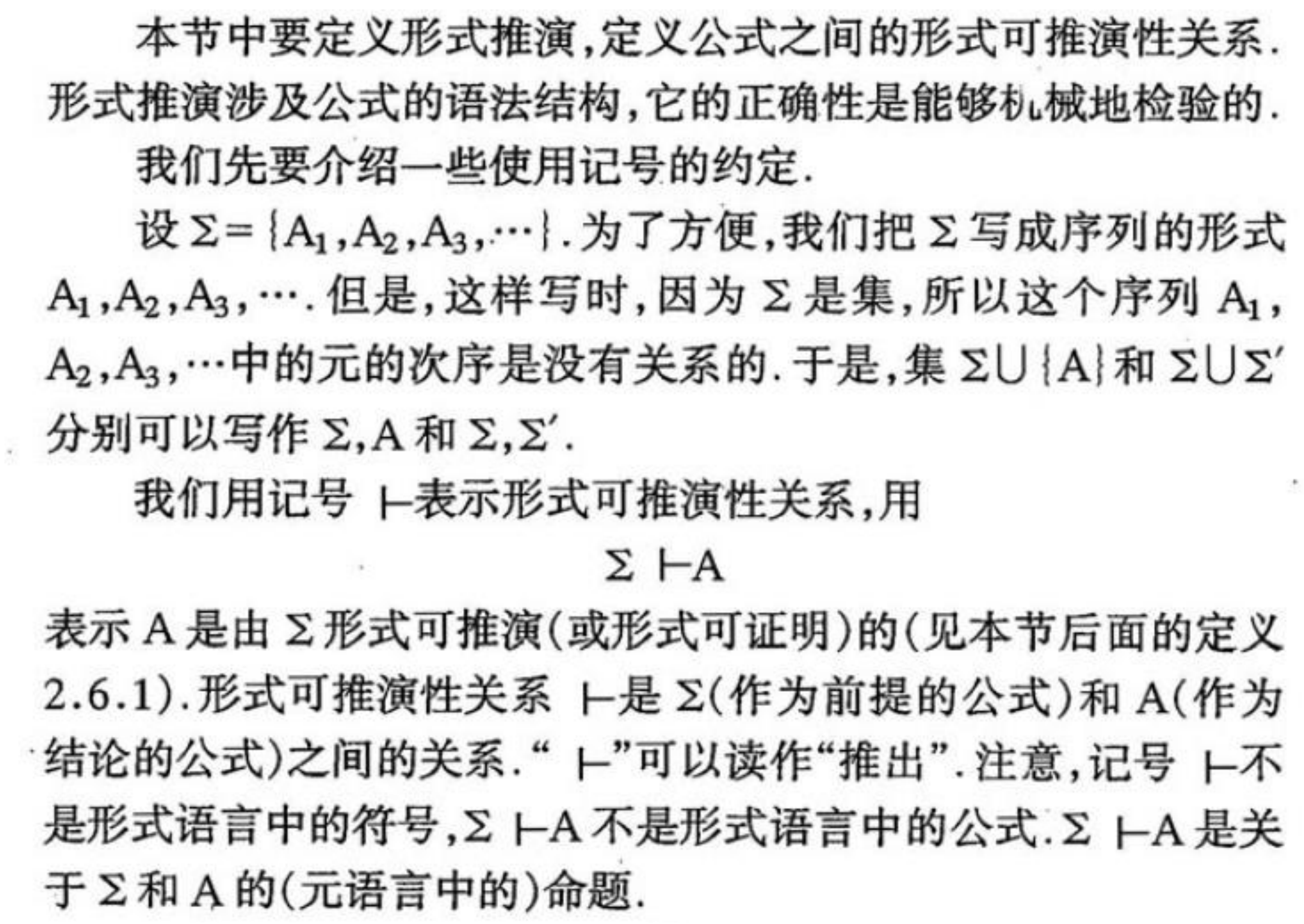
中国科学院计算技术研究所
Institute Of Computing Technology Chinese Academy Of Sciences

罗平 luop@ict.ac.cn

Knowledge 2

形式推演 DEDUCTION

形式推演 Deduction



本节中要定义形式推演，定义公式之间的形式可推演性关系．形式推演涉及公式的语法结构，它的正确性是能够机械地检验的．

我们先要介绍一些使用记号的约定．

设 $\Sigma = \{A_1, A_2, A_3, \cdots\}$．为了方便，我们把 Σ 写成序列的形式 $A_1, A_2, A_3, \cdots$．但是，这样写时，因为 Σ 是集，所以这个序列 $A_1, A_2, A_3, \cdots$ 中的元的次序是没有关系的．于是，集 $\Sigma \cup \{A\}$ 和 $\Sigma \cup \Sigma'$ 分别可以写作 Σ, A 和 Σ, Σ'．

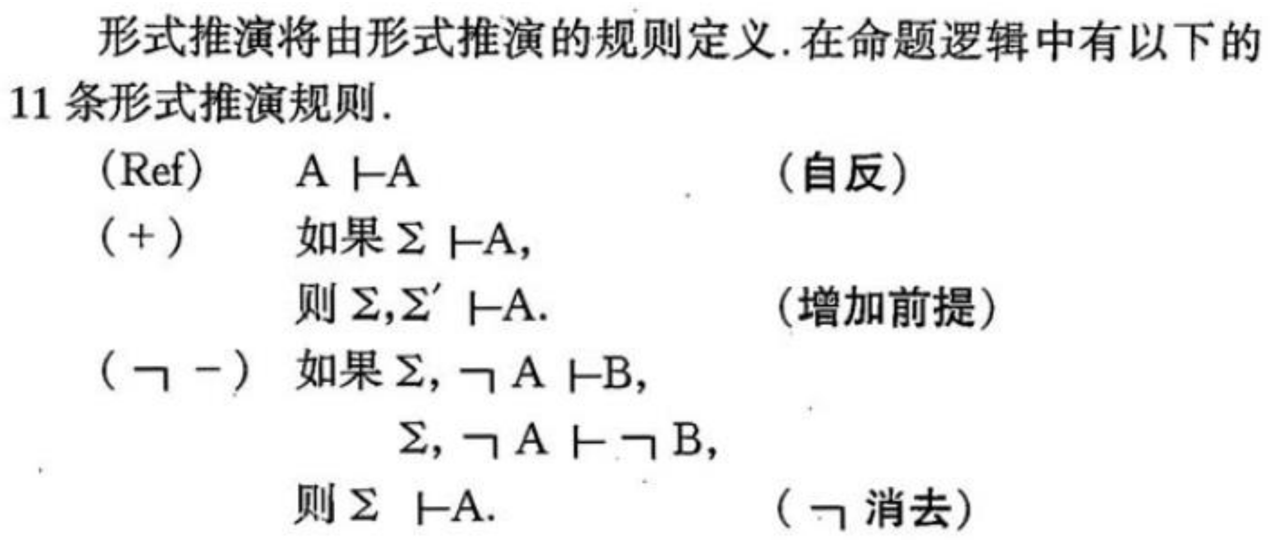
我们用记号 $\vdash$ 表示形式可推演性关系，用

$$\Sigma \vdash A$$

表示 A 是由 Σ 形式可推演（或形式可证明）的（见本节后面的定义 2.6.1）．形式可推演性关系 $\vdash$ 是 Σ（作为前提的公式）和 A（作为结论的公式）之间的关系．“$\vdash$”可以读作“推出”．注意，记号 $\vdash$ 不是形式语言中的符号，$\Sigma \vdash A$ 不是形式语言中的公式．$\Sigma \vdash A$ 是关于 Σ 和 A 的（元语言中的）命题．

《面向计算机科学的数理逻辑》49页

形式推演的11条规则（某一种推演系统）



形式推演将由形式推演的规则定义. 在命题逻辑中有以下的 11 条形式推演规则.

(Ref) $A \vdash A$ (自反)

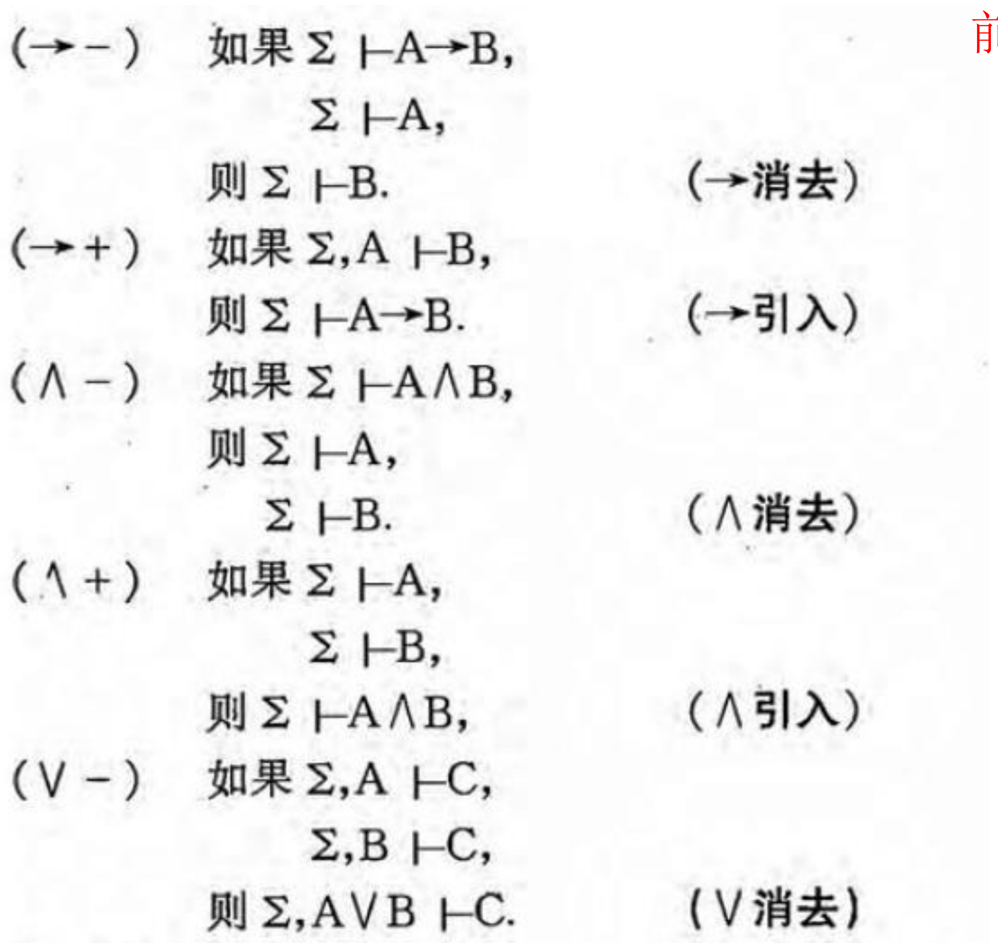
(+) 如果 $\Sigma \vdash A$,
则 $\Sigma, \Sigma' \vdash A$. (增加前提)

($\neg$ −) 如果 $\Sigma, \neg A \vdash B$,
$\Sigma, \neg A \vdash \neg B$,
则 $\Sigma \vdash A$. ($\neg$ 消去)

《面向计算机科学的数理逻辑》49页

形式推演的11条规则

符号提示：
这里的→ 就是
前面定义的⇒



($\to -$) 如果 $\Sigma \vdash A \to B$,
$\Sigma \vdash A$,
则 $\Sigma \vdash B$. ($\to$消去)

($\to +$) 如果 $\Sigma, A \vdash B$,
则 $\Sigma \vdash A \to B$. ($\to$引入)

($\wedge -$) 如果 $\Sigma \vdash A \wedge B$,
则 $\Sigma \vdash A$,
$\Sigma \vdash B$. ($\wedge$消去)

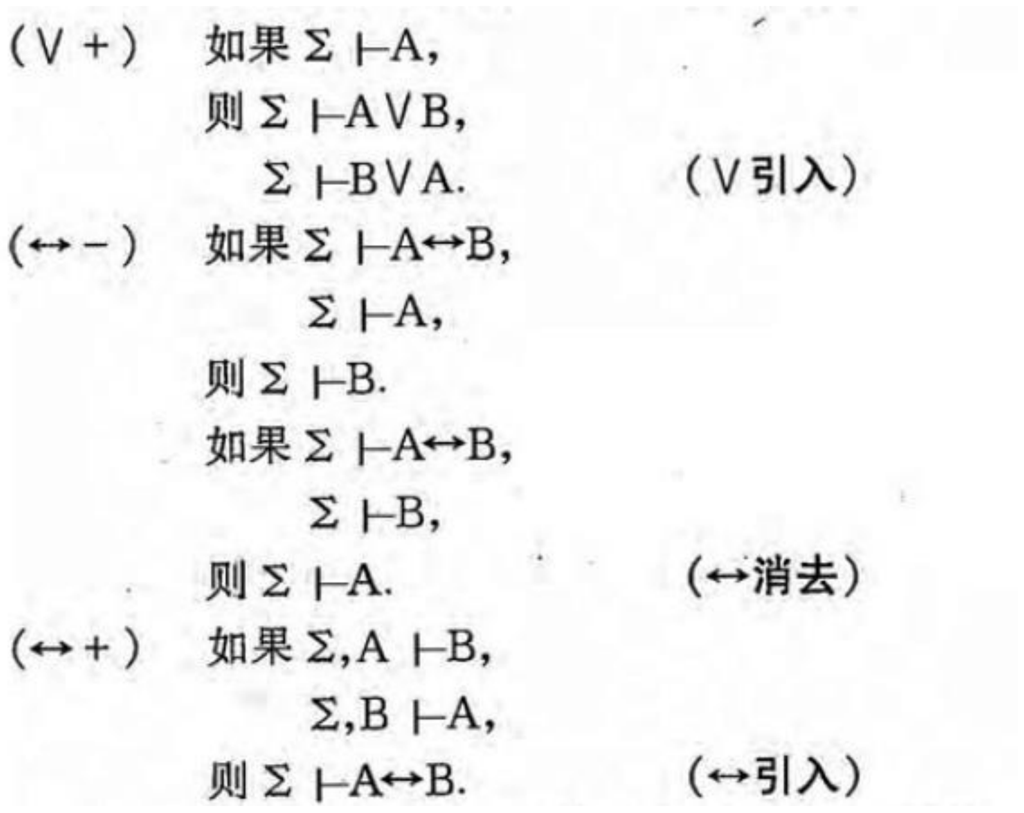
($\wedge +$) 如果 $\Sigma \vdash A$,
$\Sigma \vdash B$,
则 $\Sigma \vdash A \wedge B$, ($\wedge$引入)

($\vee -$) 如果 $\Sigma, A \vdash C$,
$\Sigma, B \vdash C$,
则 $\Sigma, A \vee B \vdash C$. ($\vee$消去)

《面向计算机科学的数理逻辑》49页

形式推演的11条规则

符号提示：
这里的↔ 就是
前面定义的⇔

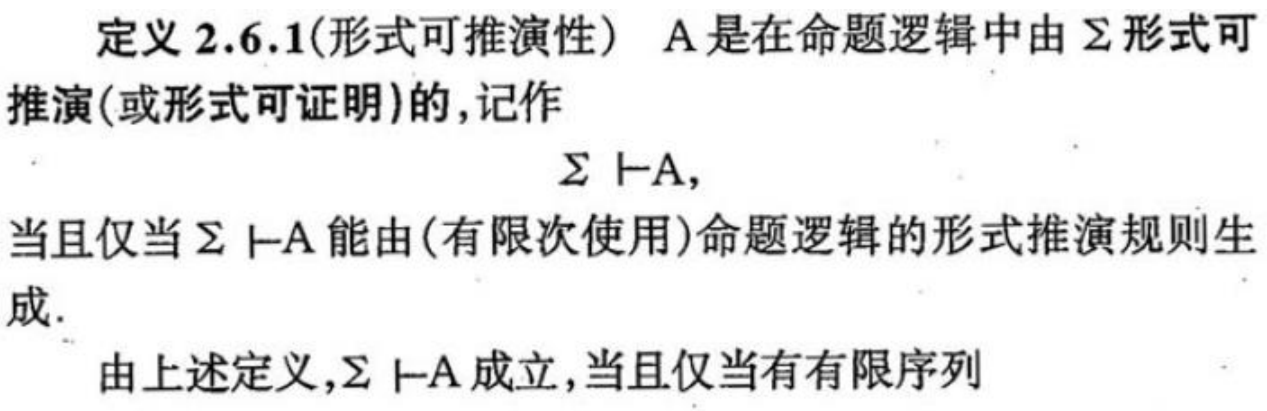


(∨+) 如果 $\Sigma \vdash A$,
则 $\Sigma \vdash A \vee B$,
$\Sigma \vdash B \vee A$. (∨引入)

(↔−) 如果 $\Sigma \vdash A \leftrightarrow B$,
$\Sigma \vdash A$,
则 $\Sigma \vdash B$.
如果 $\Sigma \vdash A \leftrightarrow B$,
$\Sigma \vdash B$,
则 $\Sigma \vdash A$. (↔消去)

(↔+) 如果 $\Sigma, A \vdash B$,
$\Sigma, B \vdash A$,
则 $\Sigma \vdash A \leftrightarrow B$. (↔引入)

形式推演

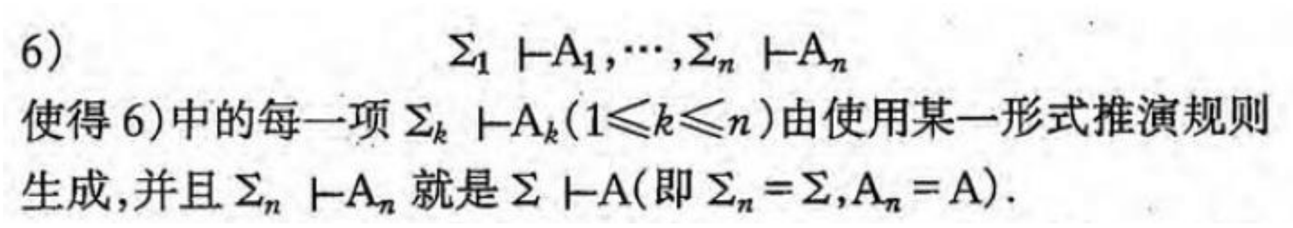


定义 2.6.1(形式可推演性)　A 是在命题逻辑中由 Σ 形式可推演(或形式可证明)的,记作

$$\Sigma \vdash A,$$

当且仅当 $\Sigma \vdash A$ 能由(有限次使用)命题逻辑的形式推演规则生成.

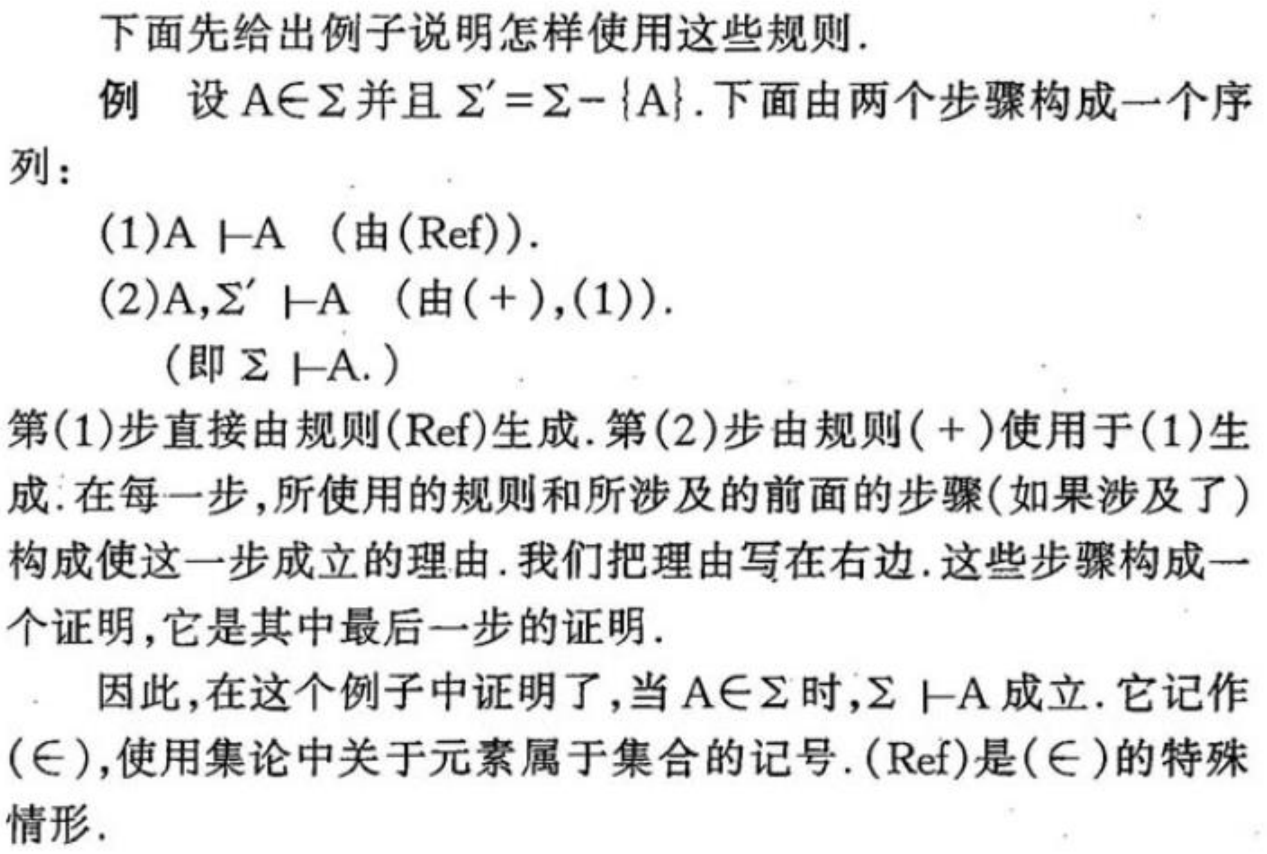
由上述定义,$\Sigma \vdash A$ 成立,当且仅当有有限序列



$$\Sigma_1 \vdash A_1, \cdots, \Sigma_n \vdash A_n \tag{6}$$

使得 6)中的每一项 $\Sigma_k \vdash A_k$($1 \leqslant k \leqslant n$)由使用某一形式推演规则生成,并且 $\Sigma_n \vdash A_n$ 就是 $\Sigma \vdash A$(即 $\Sigma_n = \Sigma, A_n = A$).

形式推演：例子



下面先给出例子说明怎样使用这些规则.

例 设 $A\in\Sigma$ 并且 $\Sigma'=\Sigma-\{A\}$. 下面由两个步骤构成一个序列:

(1) $A \vdash A$ (由(Ref)).

(2) $A,\Sigma' \vdash A$ (由(+),(1)).

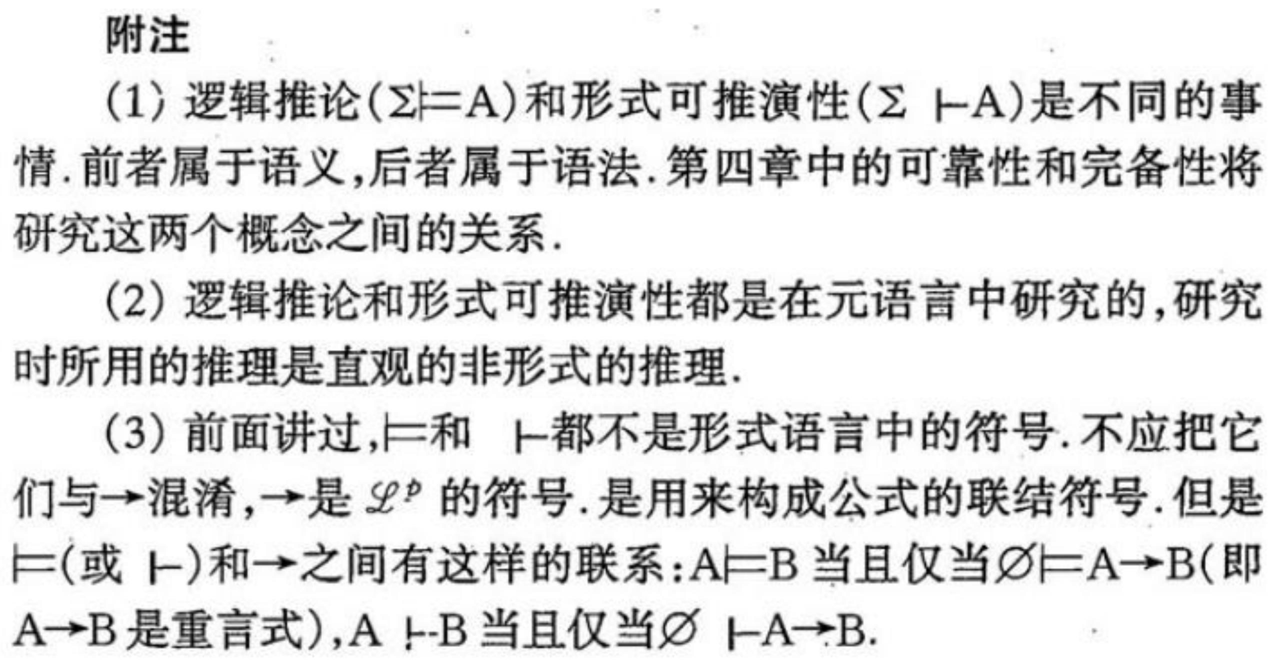
(即 $\Sigma \vdash A$.)

第(1)步直接由规则(Ref)生成. 第(2)步由规则(+)使用于(1)生成. 在每一步,所使用的规则和所涉及的前面的步骤(如果涉及了)构成使这一步成立的理由. 我们把理由写在右边. 这些步骤构成一个证明,它是其中最后一步的证明.

因此,在这个例子中证明了,当 $A\in\Sigma$ 时, $\Sigma \vdash A$ 成立. 它记作($\in$),使用集论中关于元素属于集合的记号. (Ref)是($\in$)的特殊情形.

《面向计算机科学的数理逻辑》51页

逻辑推导 vs. 形式推演



附注

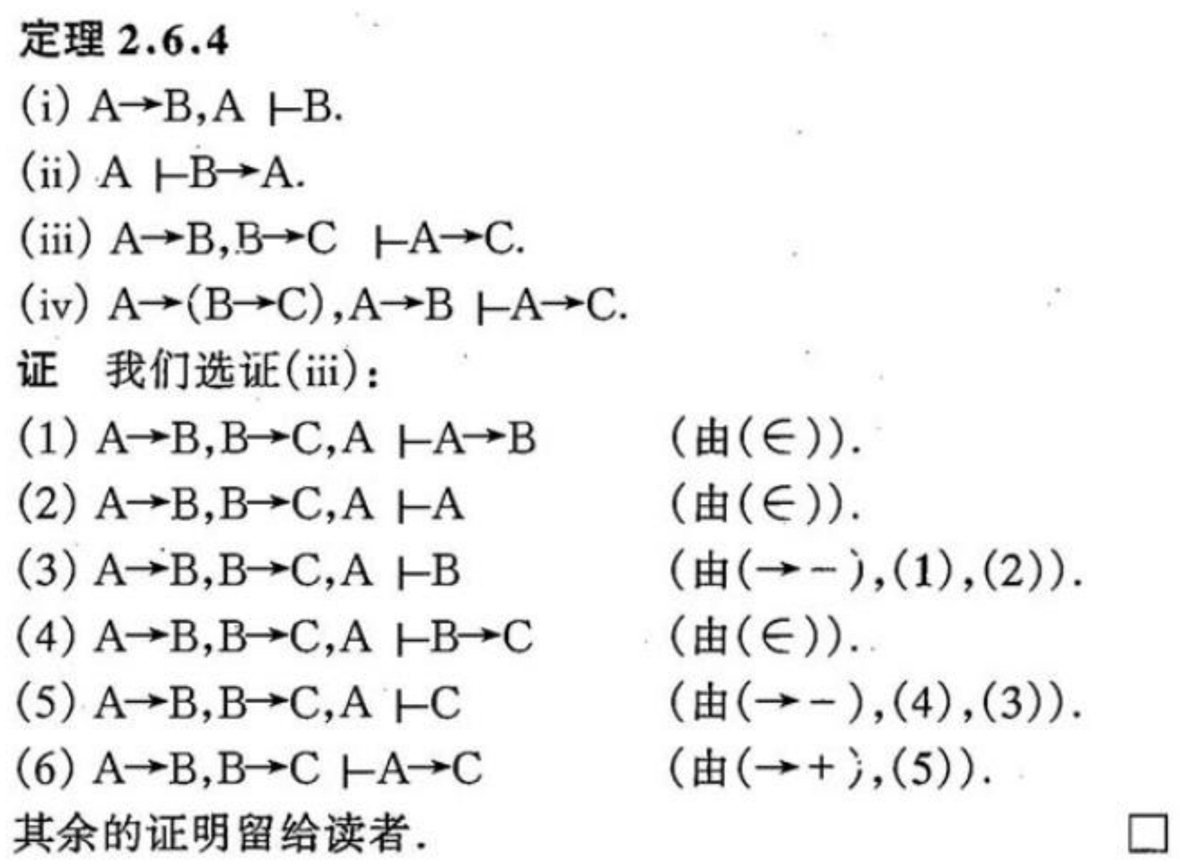
(1) 逻辑推论($\Sigma \models A$)和形式可推演性($\Sigma \vdash A$)是不同的事情.前者属于语义,后者属于语法.第四章中的可靠性和完备性将研究这两个概念之间的关系.

(2) 逻辑推论和形式可推演性都是在元语言中研究的,研究时所用的推理是直观的非形式的推理.

(3) 前面讲过,$\models$和 $\vdash$都不是形式语言中的符号.不应把它们与$\rightarrow$混淆,$\rightarrow$是 $\mathscr{L}^p$ 的符号.是用来构成公式的联结符号.但是$\models$(或 $\vdash$)和$\rightarrow$之间有这样的联系:$A \models B$ 当且仅当$\varnothing \models A \rightarrow B$(即$A \rightarrow B$ 是重言式),$A \vdash B$ 当且仅当$\varnothing \vdash A \rightarrow B$.

《面向计算机科学的数理逻辑》55页

形式推演：例子



定理 2.6.4

(i) $A \to B, A \vdash B$.

(ii) $A \vdash B \to A$.

(iii) $A \to B, B \to C \vdash A \to C$.

(iv) $A \to (B \to C), A \to B \vdash A \to C$.

证 我们选证(iii)：

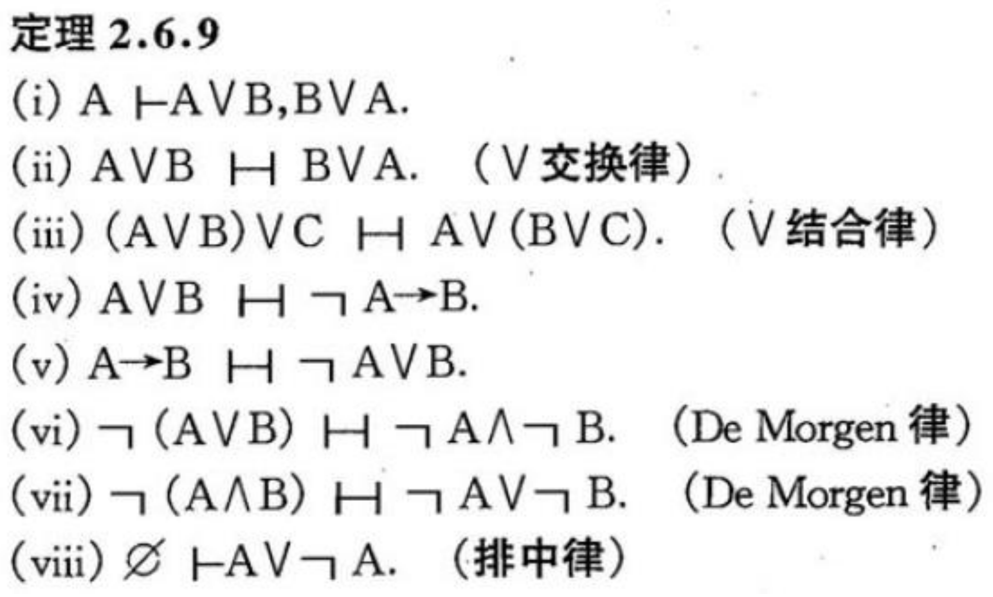
(1) $A \to B, B \to C, A \vdash A \to B$	(由($\in$)).
(2) $A \to B, B \to C, A \vdash A$	(由($\in$)).
(3) $A \to B, B \to C, A \vdash B$	(由($\to -$),(1),(2)).
(4) $A \to B, B \to C, A \vdash B \to C$	(由($\in$)).
(5) $A \to B, B \to C, A \vdash C$	(由($\to -$),(4),(3)).
(6) $A \to B, B \to C \vdash A \to C$	(由($\to +$),(5)).

其余的证明留给读者. □

Homework：其余题目

《面向计算机科学的数理逻辑》59页

形式推演：例子



定理 2.6.9

(i) $A \vdash A \vee B, B \vee A$.

(ii) $A \vee B \;\vdash\dashv\; B \vee A$. （$\vee$ **交换律**）

(iii) $(A \vee B) \vee C \;\vdash\dashv\; A \vee (B \vee C)$. （$\vee$ **结合律**）

(iv) $A \vee B \;\vdash\dashv\; \neg A \to B$.

(v) $A \to B \;\vdash\dashv\; \neg A \vee B$.

(vi) $\neg (A \vee B) \;\vdash\dashv\; \neg A \wedge \neg B$. （De Morgen **律**）

(vii) $\neg (A \wedge B) \;\vdash\dashv\; \neg A \vee \neg B$. （De Morgen **律**）

(viii) $\varnothing \vdash A \vee \neg A$. （**排中律**）

Homework：证明这些性质

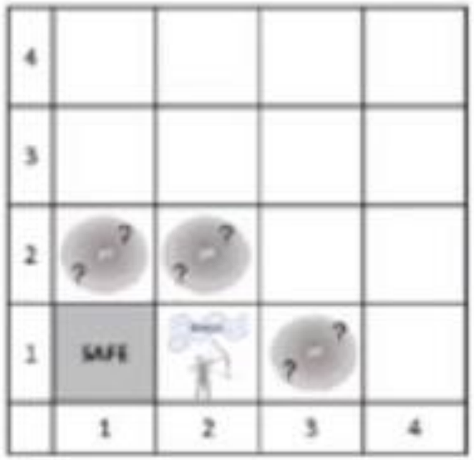
形式推演：常用的定理

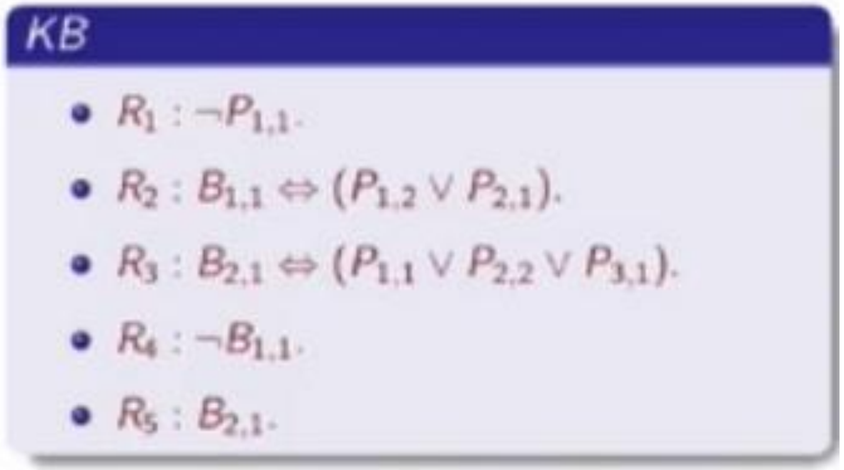
- 定理2.6.3， 2.6.4， 2.6.5， 2.6.6， 2.6.7， 2.6.8， 2.6.9， 2.6.10, 2.6.11

- 可以用基本的11条法则证明，它们则可以在其它证明中使用

形式推演：Wumpus

- $P_{x,y}$ is true if there is a pit in [x,y]
- $W_{x,y}$ is true if there is a wumpus in [x,y]
- $B_{x,y}$ is true if the agent perceives BREEZE in [x,y]
- $B_{x,y}$ is true if the agent perceives STENCH in [x,y]





KB

- $R_1 : \neg P_{1,1}$.
- $R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
- $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.
- $R_4 : \neg B_{1,1}$.
- $R_5 : B_{2,1}$.

证明：$\text{KB} \vdash \neg P_{1,2} \wedge \neg P_{2,1}$

形式推演：Wumpus

Biconditional elimination from R_2

$R_6: \left(B_{1,1} \Rightarrow (P_{1.2} \vee P_{2,1})\right) \wedge \left((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}\right)$

And – elimination from R_6

$R_7: (P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$

Contrapositive from R_7 .

$R_8: \neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$

Modus Ponens from R_8 .

$R_9: \neg(P_{1,2} \vee P_{2,1})$

De Morgan ‘s rule from R_9

$R_{10}: \neg P_{1,2} \wedge \neg P_{2,1}$

证明：$\mathrm{KB} \vdash \neg P_{1,2} \wedge \neg P_{2,1}$

证明过程并不是一个机械化的方法

Summary

- 逻辑系统
- Syntax：formal structure of sentences
- Semantics: truth of sentences wrt models; Entailment: necessary truth of one sentence given another
- Deduction: formal deduction based on deduction rules

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 $KB \vDash \alpha$

Completeness: i is complete if

whenever $KB \vDash \alpha$, 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*.

- 哥德尔不完全 定理：在一个大的范围内（证明法和问题与正整数存在一一对应关系），不存在既sound又complete的inference过程

RESOLUTION 归结原理

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

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

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

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

- Resolution is sound and complete for propositional logic

J. A. Robinson. A machine-oriented logic based on the resolution principle. **Journal of the ACM**, 1965, 12(1):23-41

Conversion to CNF

- $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- 1. Elimate$\Leftrightarrow$,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.Elimate $\Rightarrow$,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

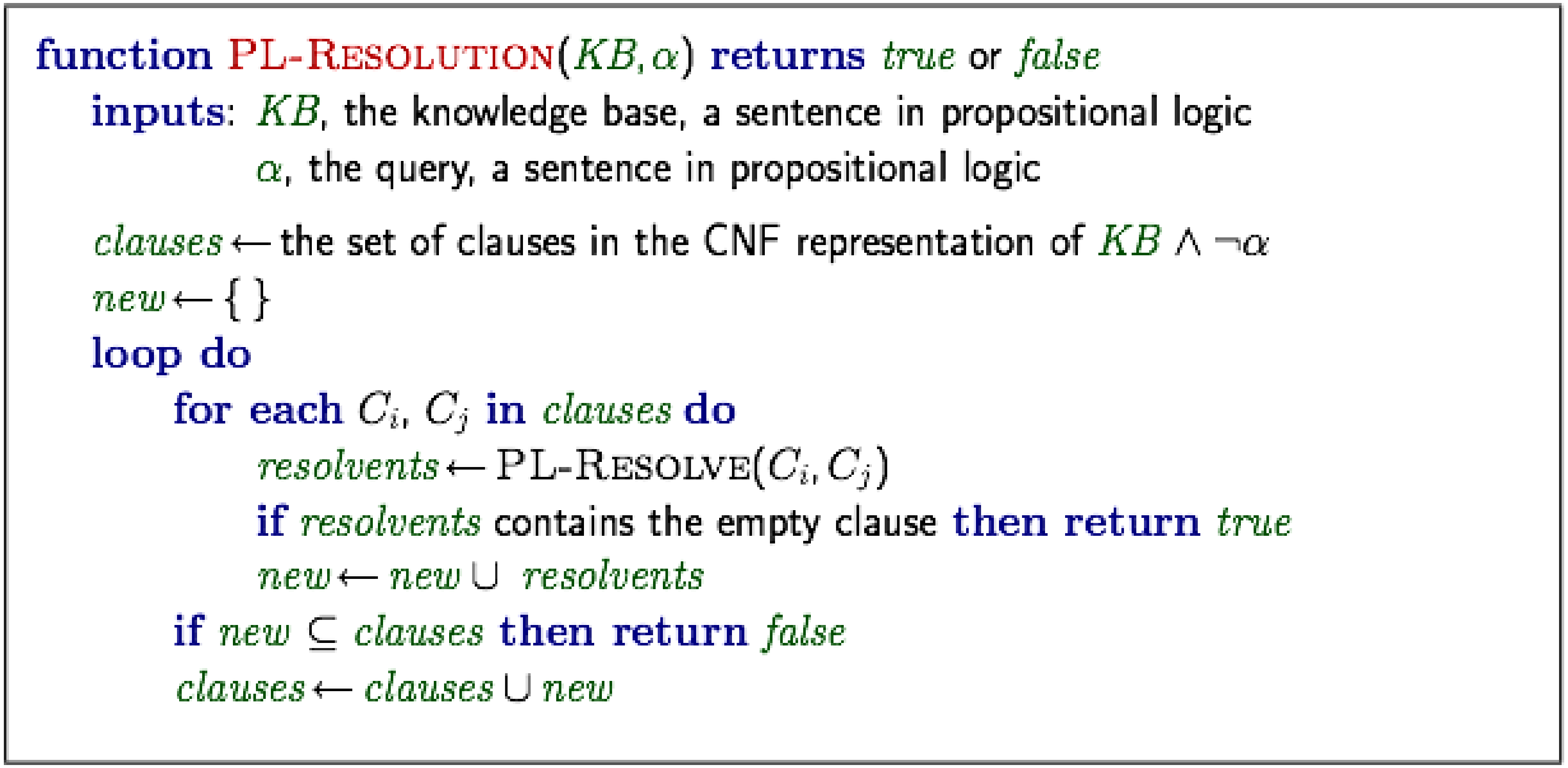
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- 4.Apply distributivity law ($\vee$ over $\wedge$) and flatten:

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

多项式时间复杂度

Resolution algorithm

- Proof by contradiction, i.e., show $KB \land \neg\alpha$ 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 ← the set of clauses in the CNF representation of $KB \land \neg\alpha$
    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

- **证明：若**$\mathrm{KB} \nvdash \emptyset$

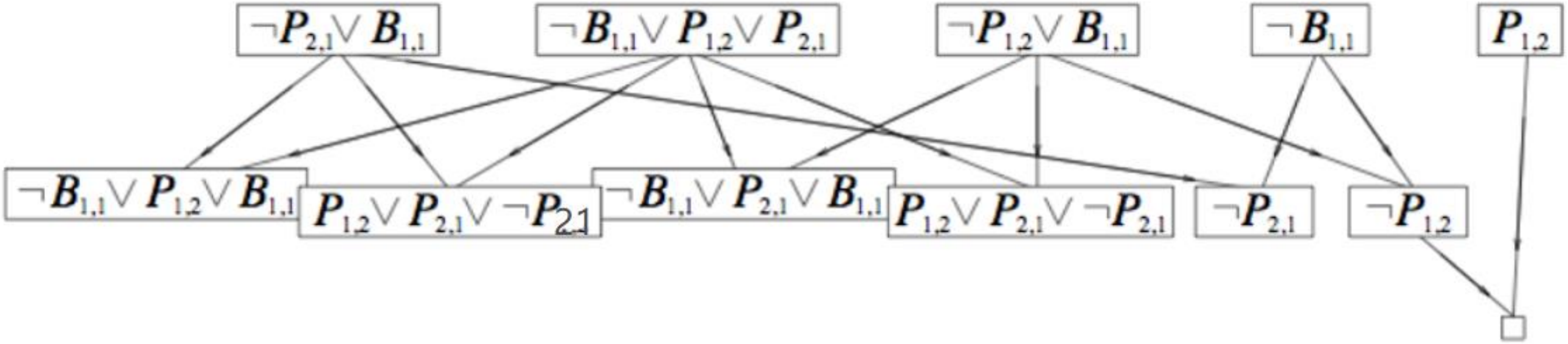
$$\mathrm{KB} \vdash \alpha \text{ 当且仅当 } \{\mathrm{KB}, \neg\alpha\} \vdash \emptyset$$

其中$\vdash$仅使用归结法则获得新子句

Resolution example

- **证明：** KB ⊢ α

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



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

Resolution is sound

- **证明：Resolution规则是可靠的。即证明：**

- $(l_1 \vee \cdots \vee l_k) \wedge (m_1 \vee \cdots \vee m_n)$
$\vDash (l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)$

Resolution规则：

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

where ℓ_i and m_j are complementary literals.

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 and all the sentences in *S*
  - 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.

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

$$KB \vDash \alpha$$

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$R_1, R_2, \cdots, R_l$，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。

从$i = 1$ 到 l，顺序的指派$R_1, R_2, \cdots, R_l$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都已经被指派为False（在之前的步骤中进行的）或不包含其它文字，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：这个真值指派使得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，且假设这是首次出现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是不会在此真值指派中为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。

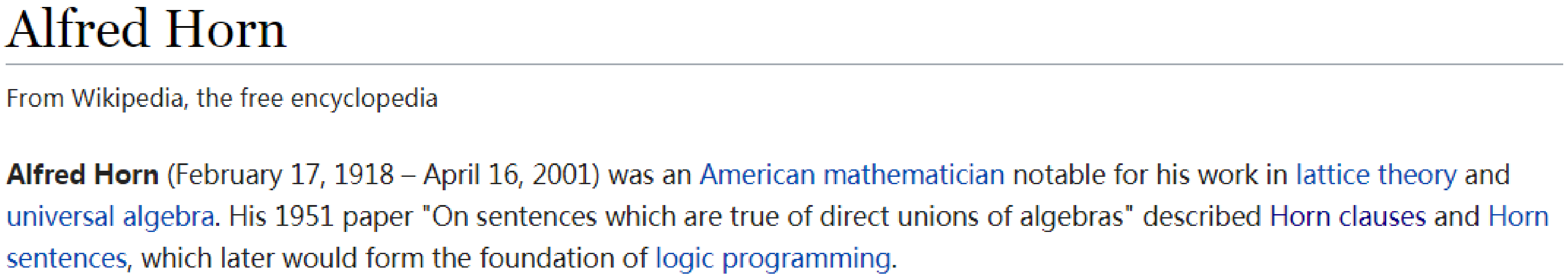
Process of Resolution: Search

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 the states are (single clause or a set of clauses (preferred))

Inference over 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

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

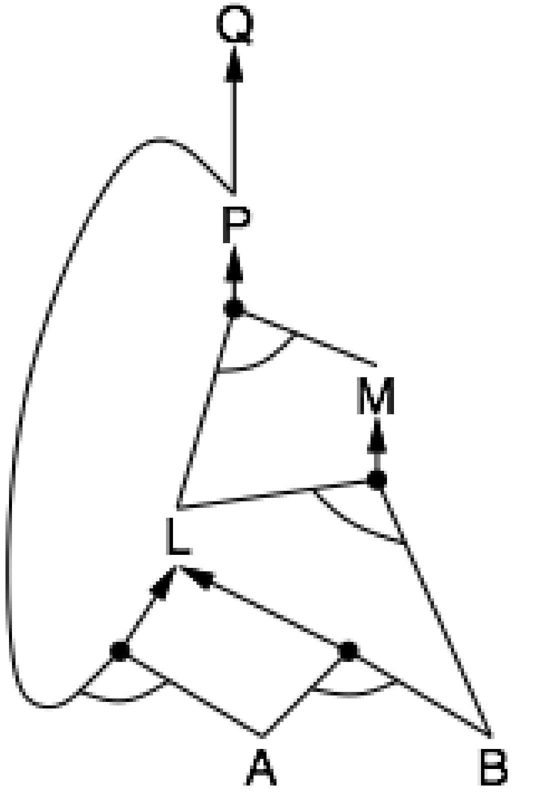
  归结的一种形式

- 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

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



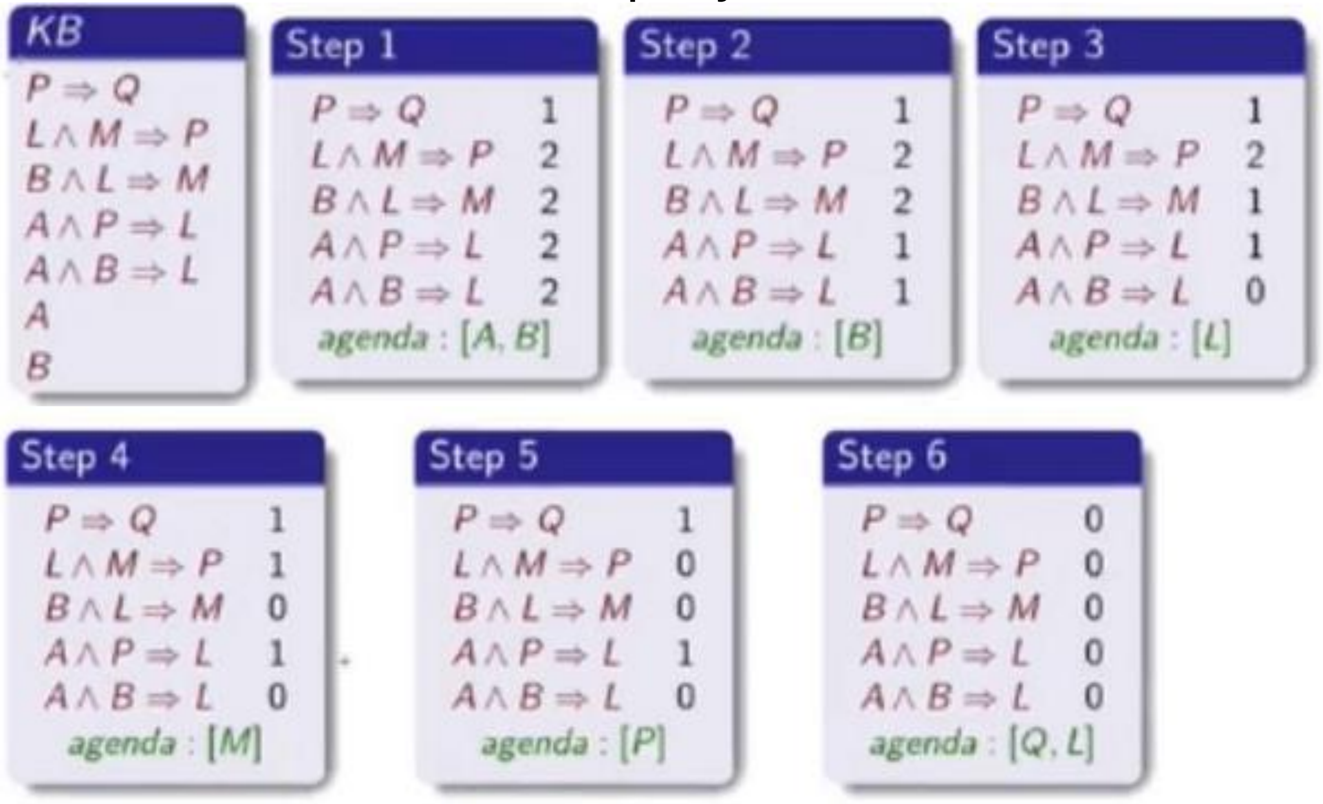
Forward chaining algorithm

```
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
       $p \leftarrow$ POP($agenda$)
       unless $inferred[p]$ do
           $inferred[p] \leftarrow true$
           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

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

Step 1	
$P \Rightarrow Q$	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
agenda : [A, B]	

Step 2	
$P \Rightarrow Q$	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
agenda : [B]	

Step 3	
$P \Rightarrow Q$	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
agenda : [L]	

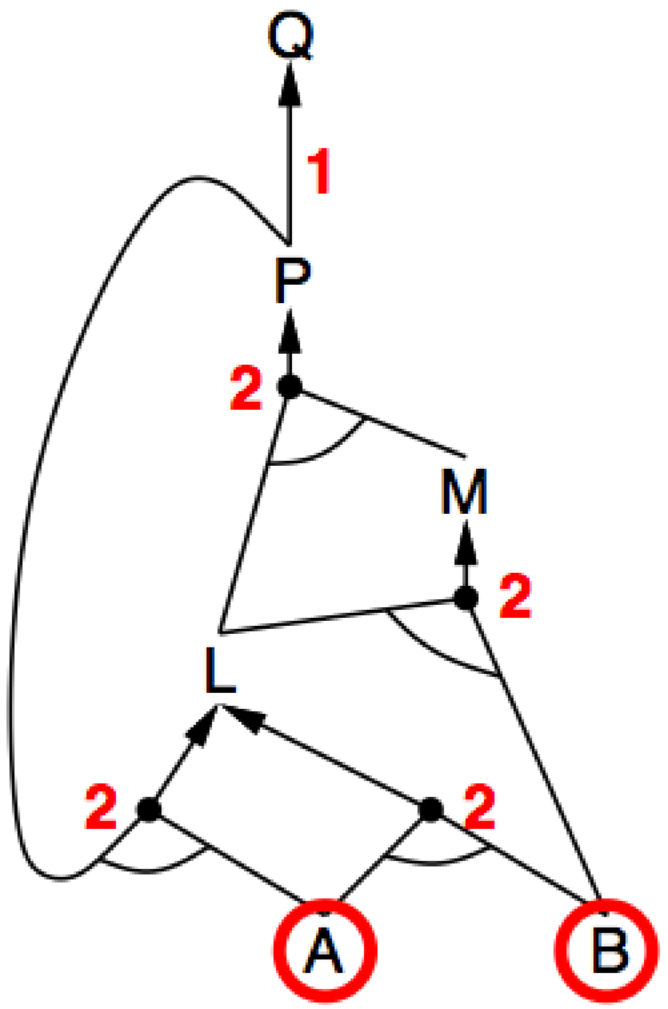
Step 4	
$P \Rightarrow Q$	1
$L \wedge M \Rightarrow P$	1
$B \wedge L \Rightarrow M$	0
$A \wedge P \Rightarrow L$	1
$A \wedge B \Rightarrow L$	0
agenda : [M]	

Step 5	
$P \Rightarrow Q$	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
agenda : [P]	

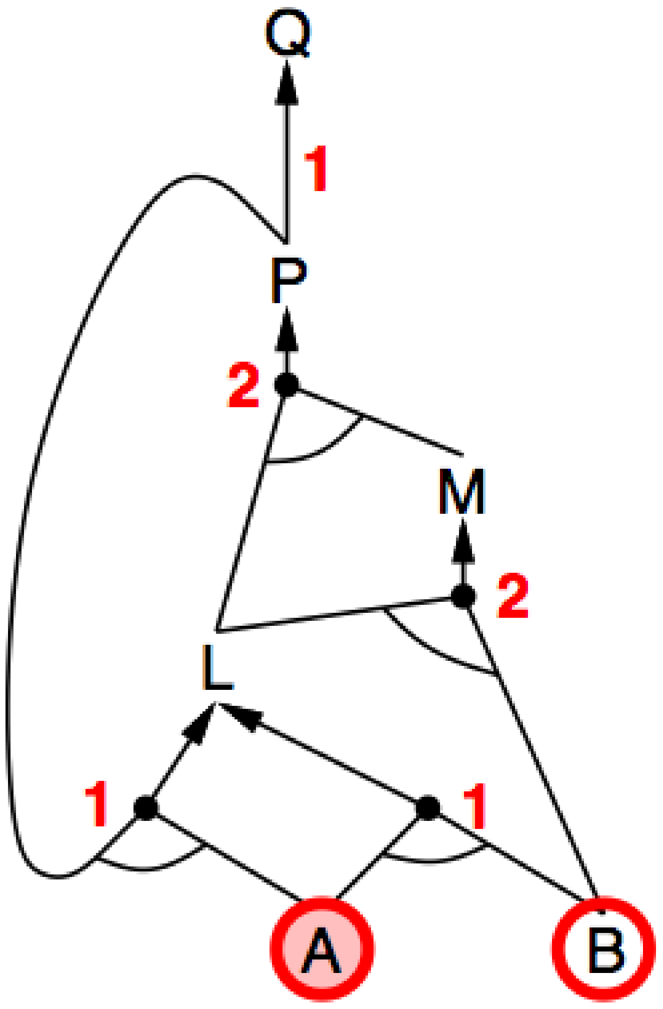
Step 6	
$P \Rightarrow Q$	0
$L \wedge M \Rightarrow P$	0
$B \wedge L \Rightarrow M$	0
$A \wedge P \Rightarrow L$	0
$A \wedge B \Rightarrow L$	0
agenda : [Q, L]	

Linear to the number of what?

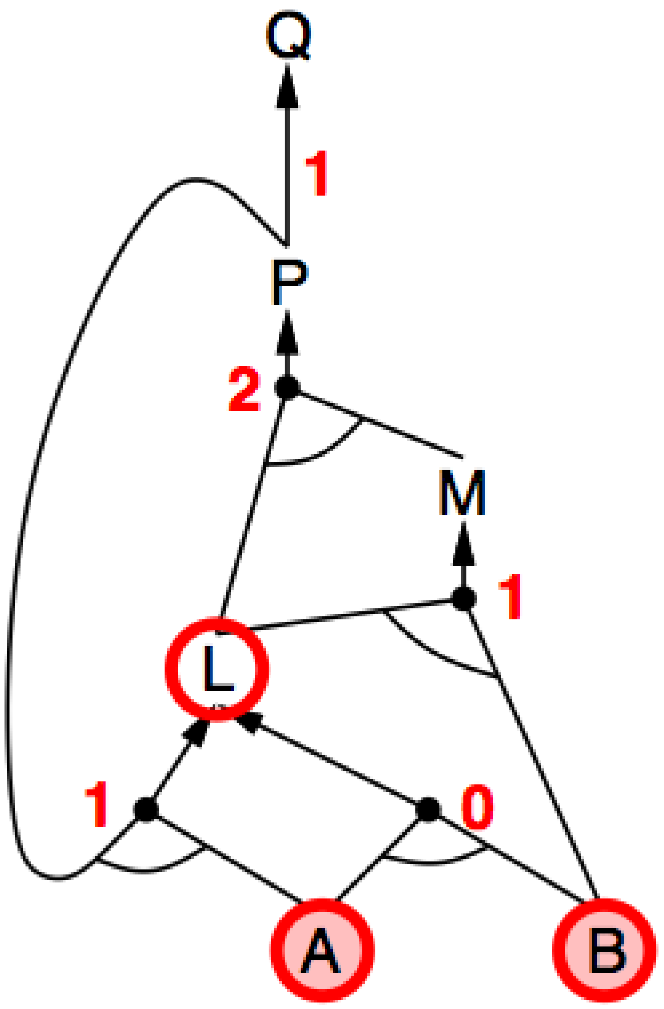
Forward chaining example



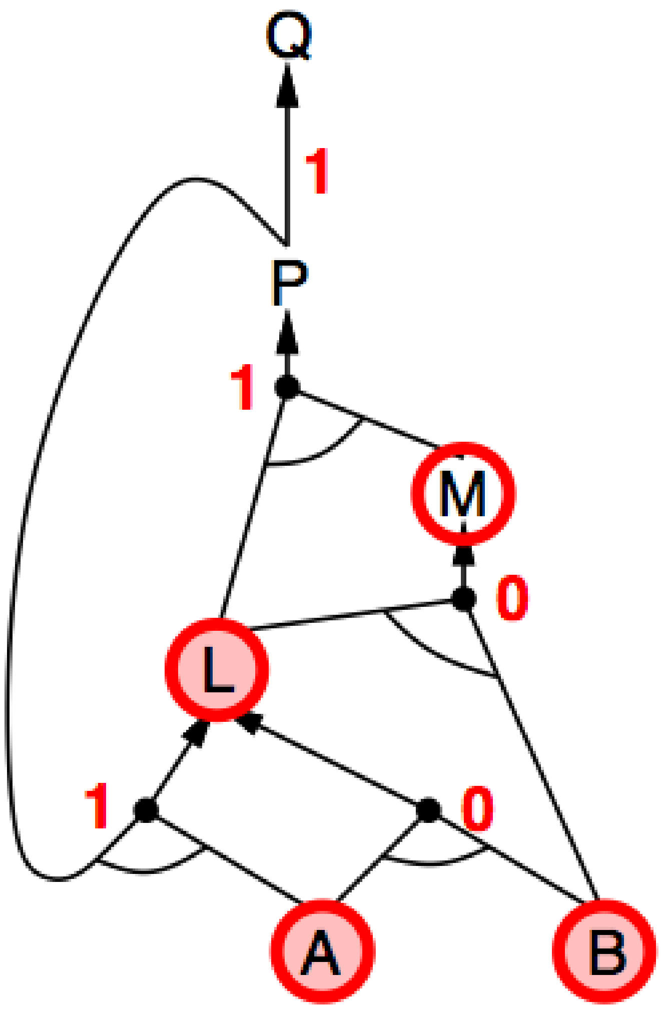
Forward chaining example



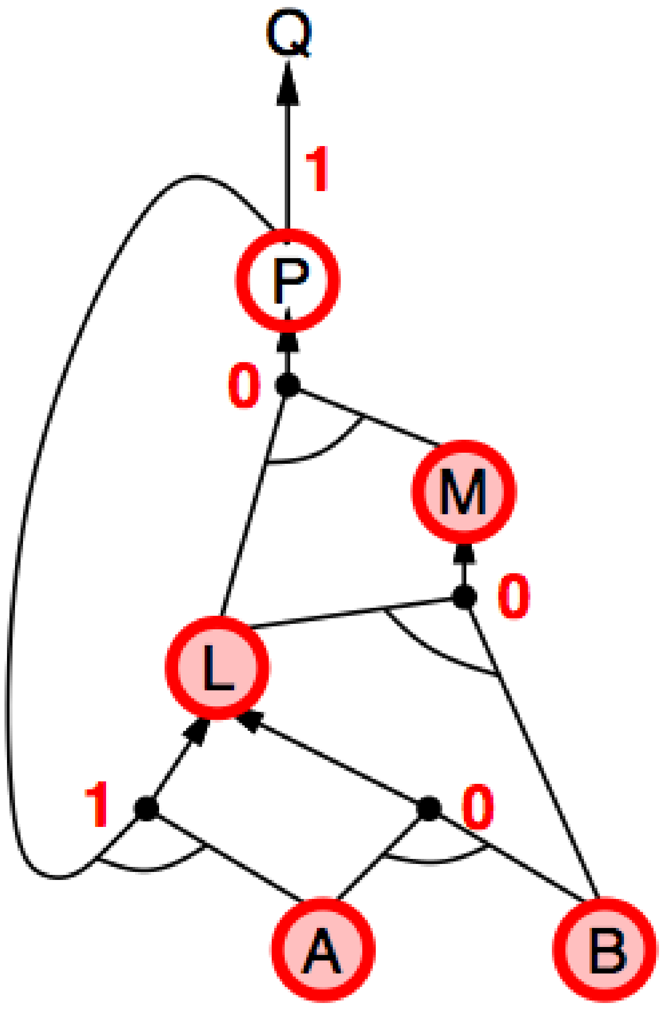
Forward chaining example



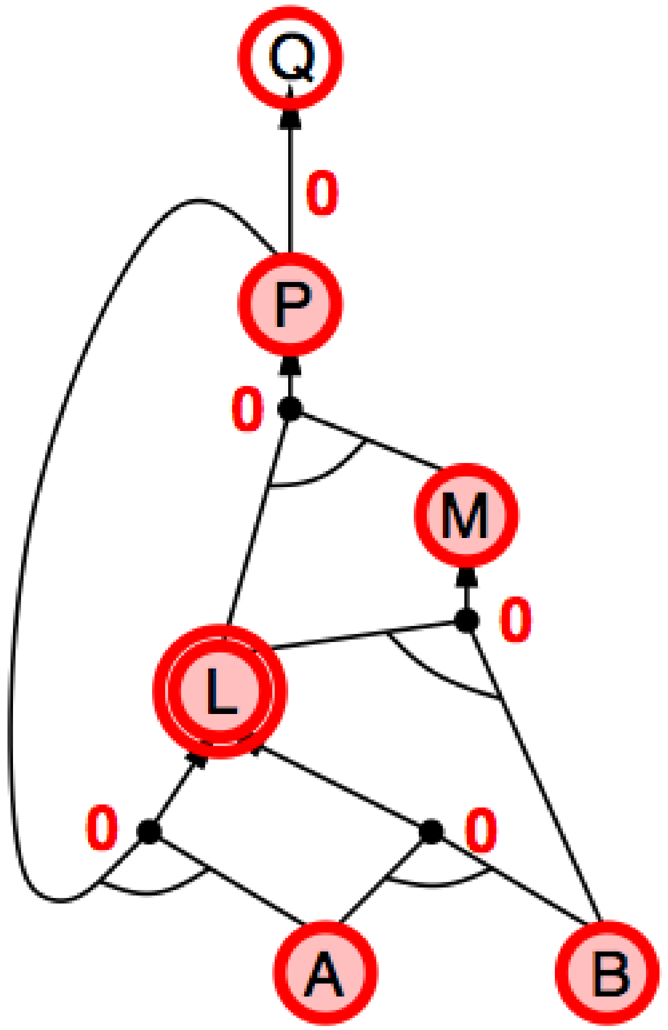
Forward chaining example



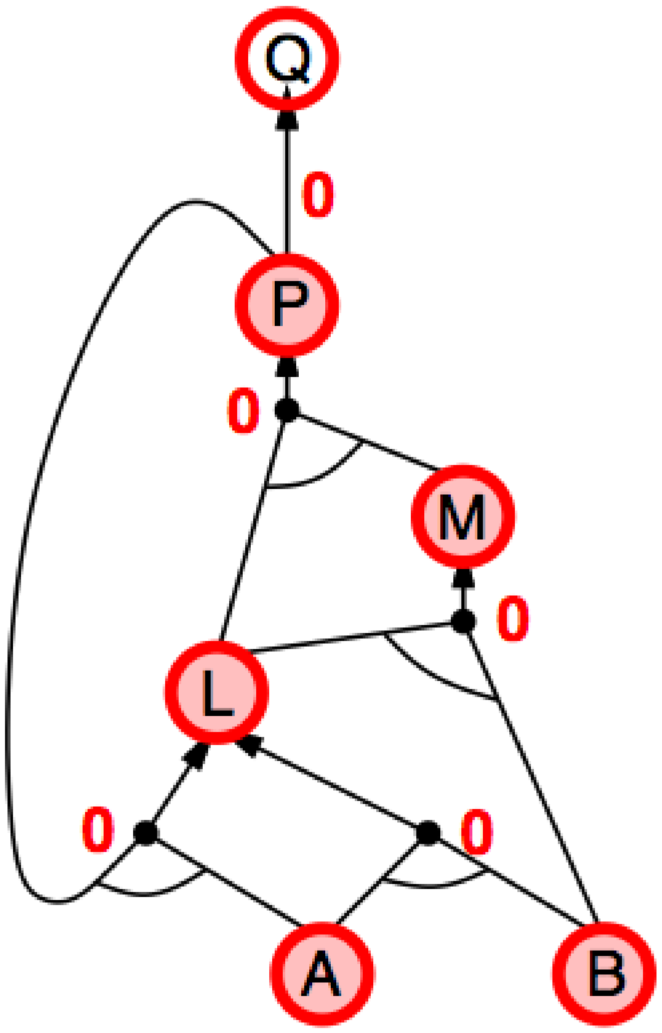
Forward chaining example



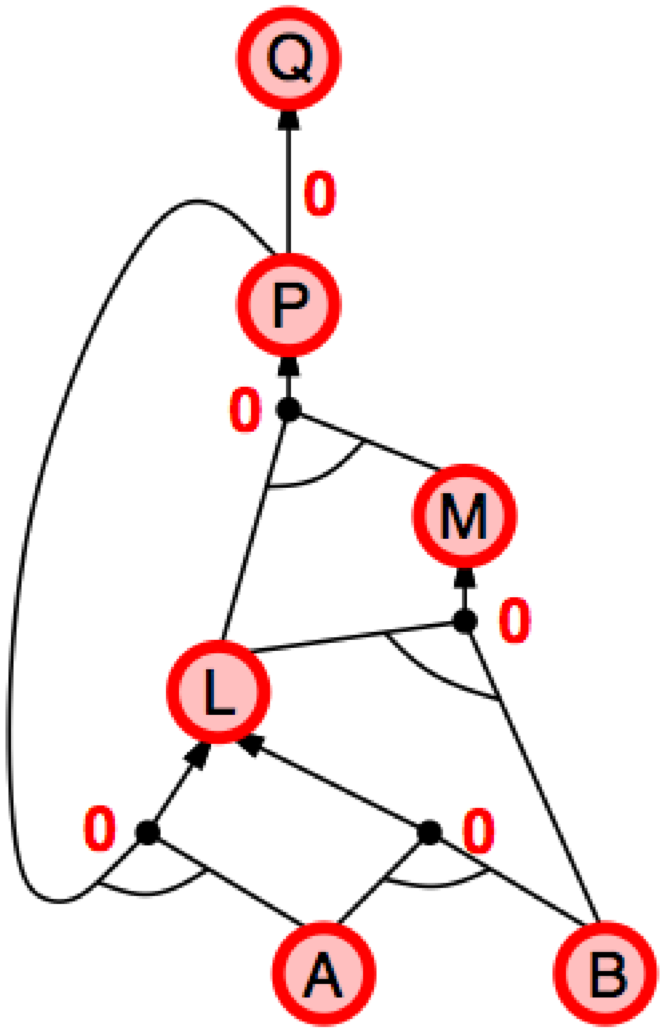
Forward chaining example



Forward chaining example



Forward chaining example

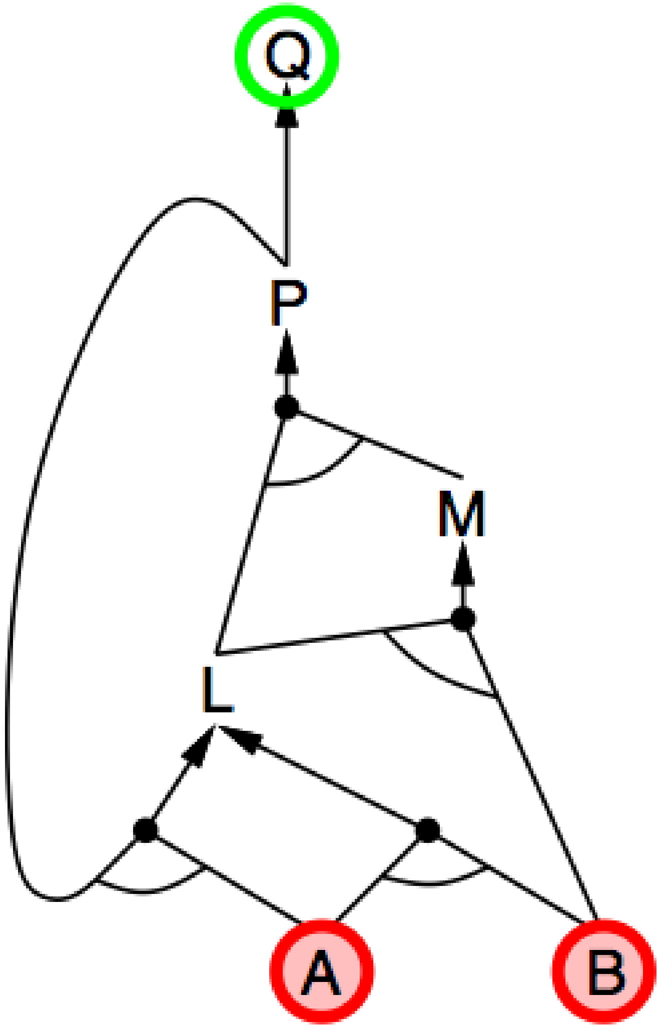


Backward chaining（后向推理）

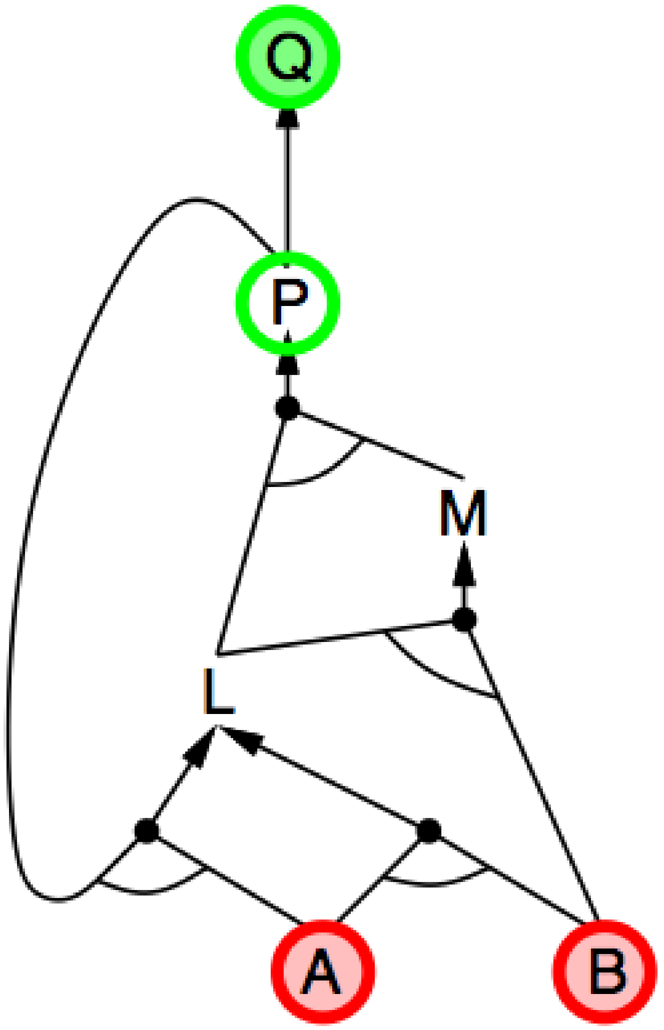
- Idea: work backwards from the query *q*:

  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

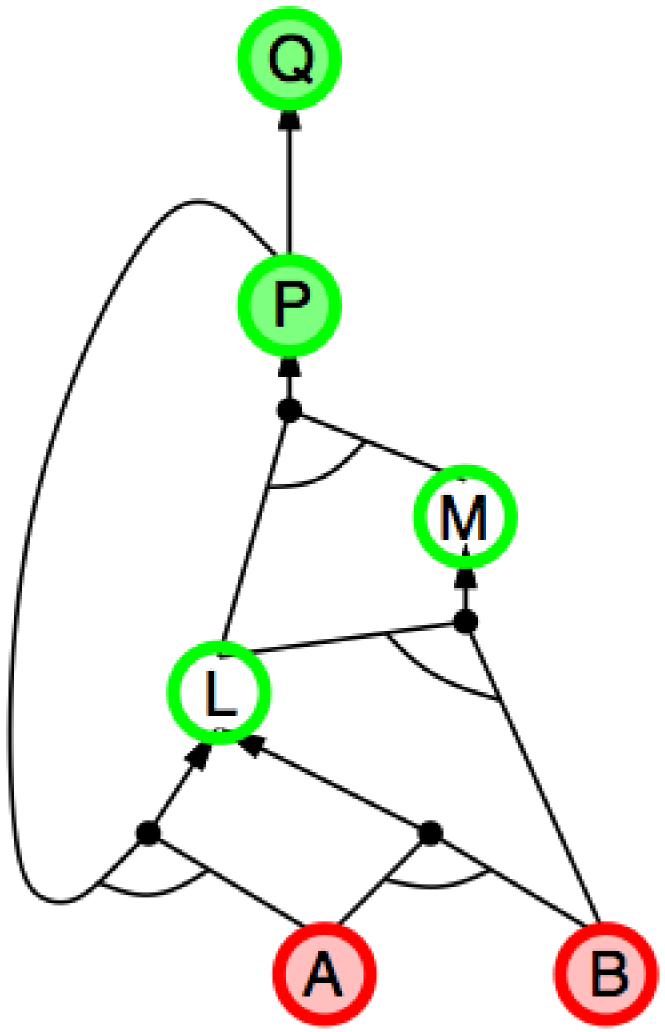
Backward chaining example



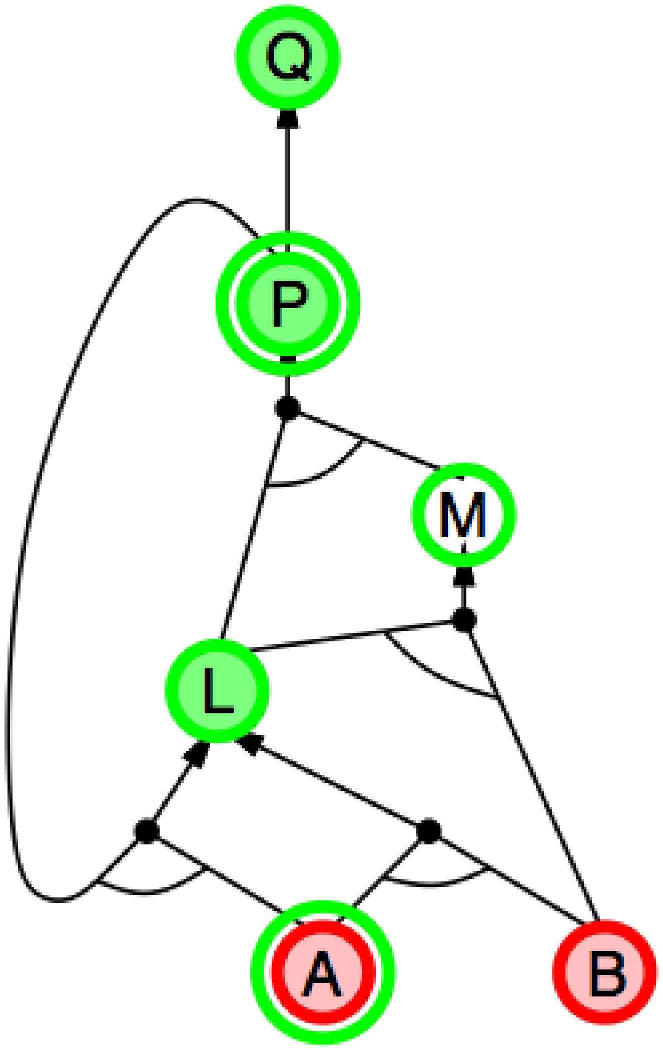
Backward chaining example



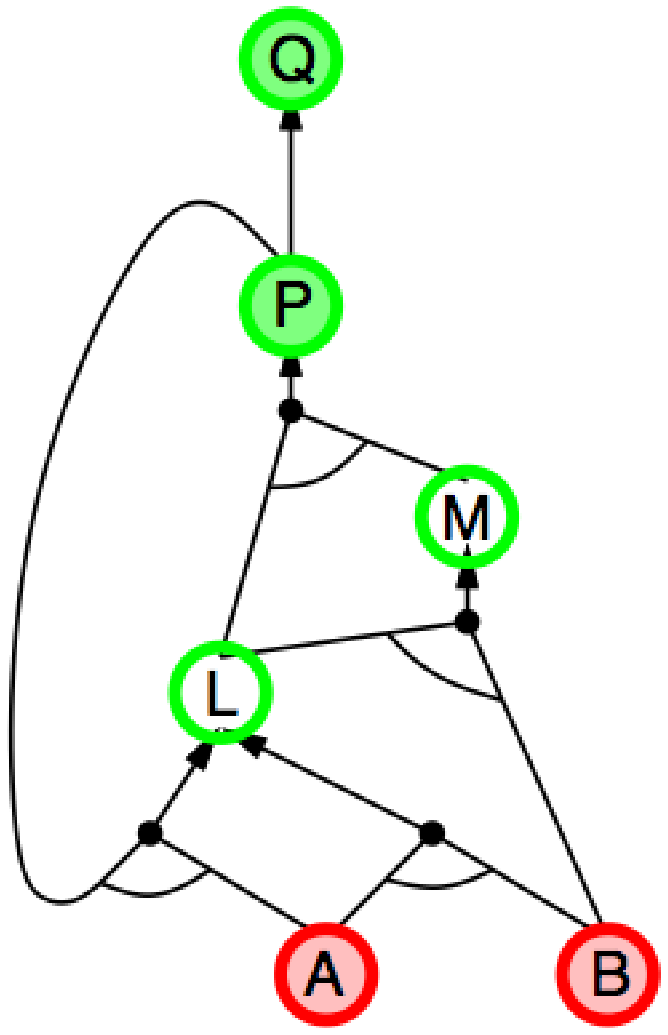
Backward chaining example



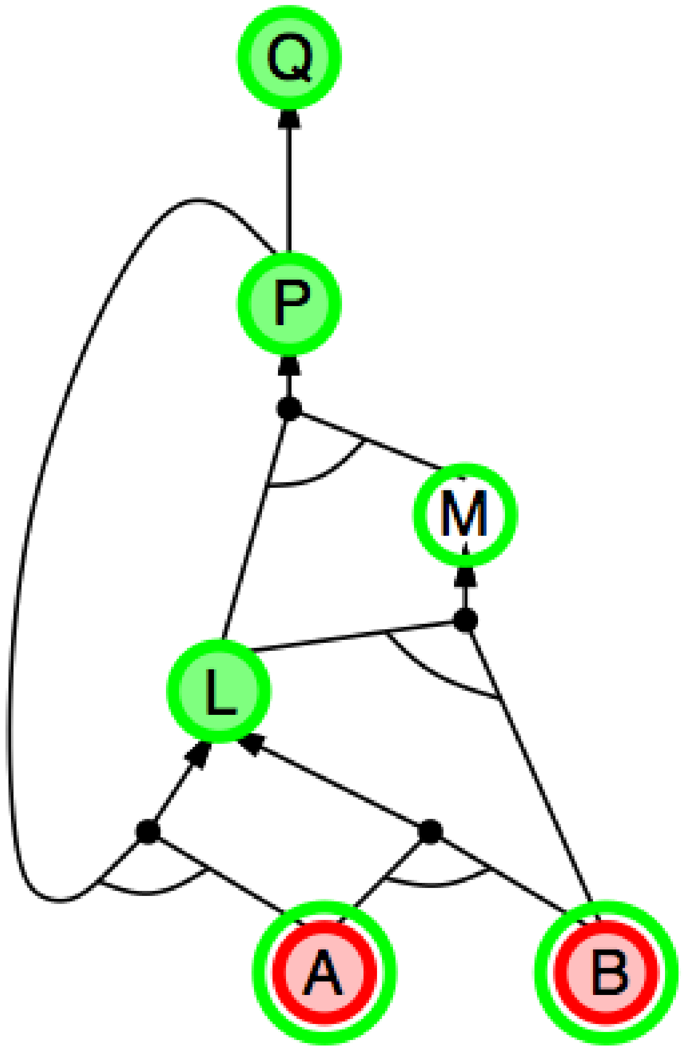
Backward chaining example



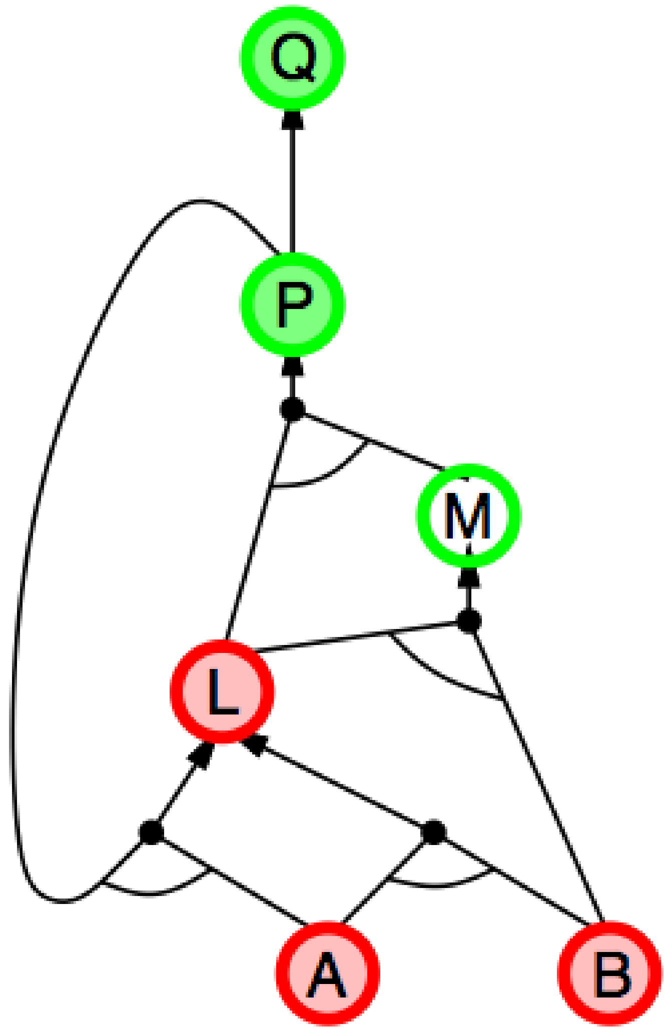
Backward chaining example



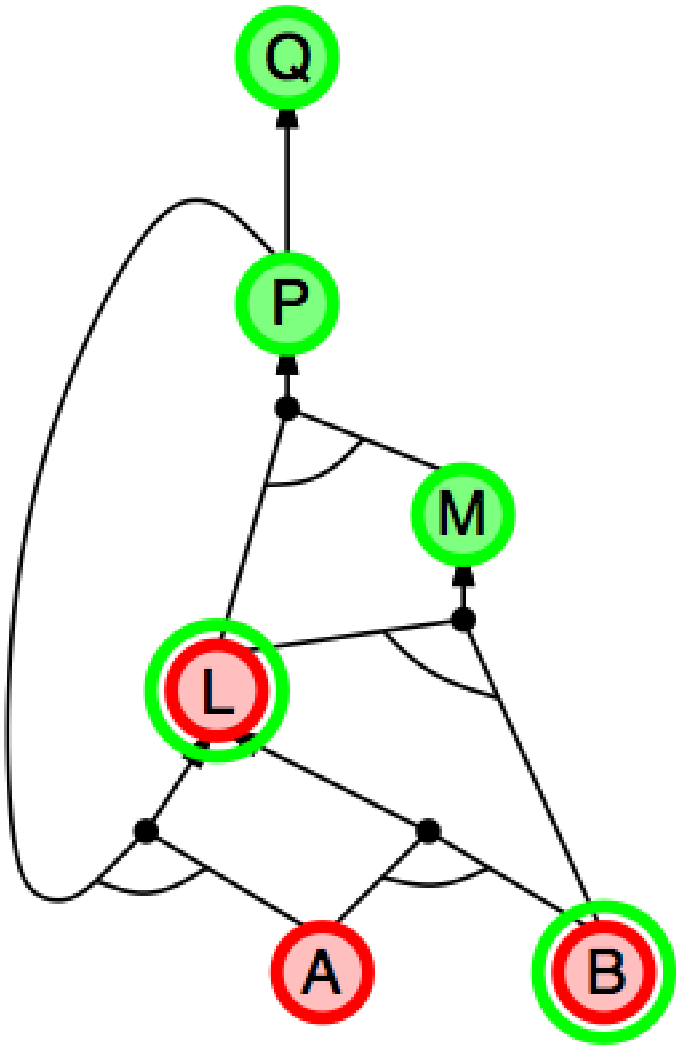
Backward chaining example



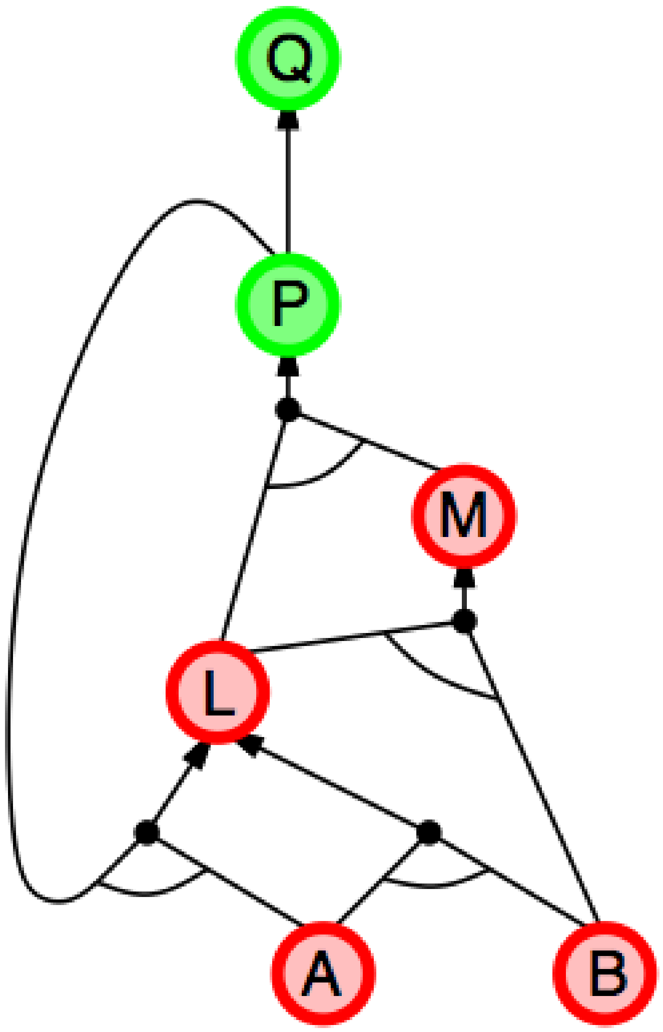
Backward chaining example



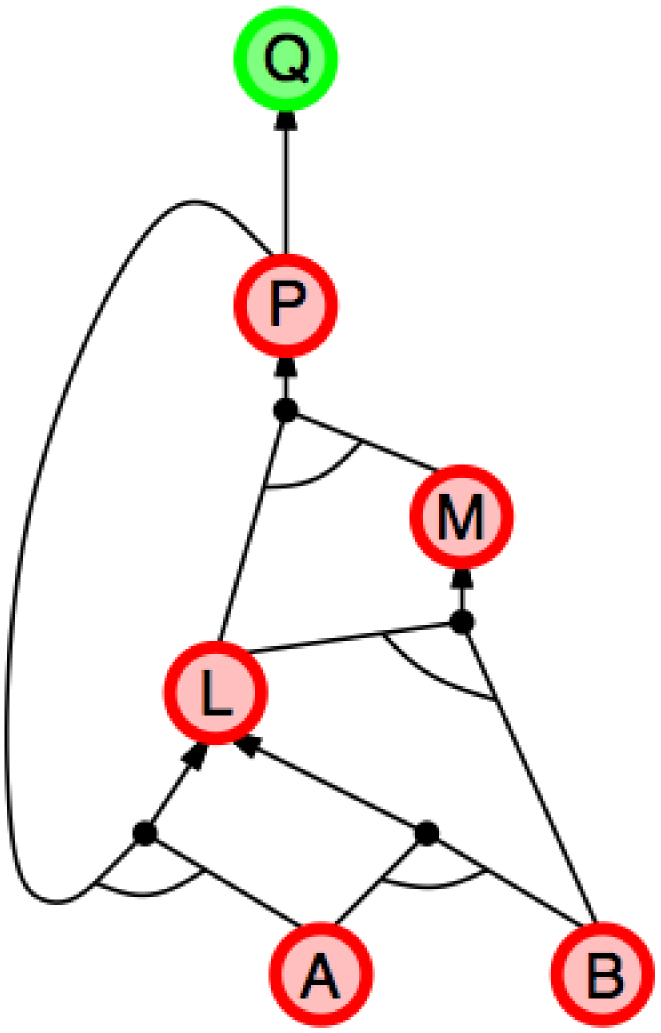
Backward chaining example



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：**

$$\alpha_1 \land \cdots \land \alpha_n \land (\alpha_1 \land \cdots \land \alpha_n \Rightarrow \beta) \vDash \beta$$

Modus Ponens规则：

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

Proof of completeness

- **若$KB \vDash \alpha$，$KB \vdash \alpha$。此时， KB中仅包含definite子句， $\vdash$仅使用Modus Ponens规则，且α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 $a_1 \wedge \cdots \wedge a_k \Rightarrow b$

  也就是说，在m中， $a_1, \cdots a_k$ 均为True，且b为False。

  根据1)中的定义， $a_i \in RC(KB)$ (i=1,⋯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$KB \vDash \alpha$，根据蕴含的定义：在m中，α为真；

  则根据1)中的定义，$\alpha \in RC(KB)$

  也就是说： $KB \vdash \alpha$

Summary on Propositional Logic

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 definite clauses

Resolution is complete for propositional logic.

Propositional logic lacks expressive power

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

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.