Lecture Notes on Inference Methods,
University of Birzeit, Palestine
Spring Semester, 2014

Artificial Intelligence

Section 7.5 (& extra)

Inference Methods
In Propositional Logic

Dr. Mustafa Jarrar

Sina Institute, University of Birzeit

mjarrar@birzeit.edu
www.jarrar.info

Mustafa Jarrar: Online Courses – Watch and Download Slides

Reference:
Mustafa Jarrar: Lecture Notes on Conceptual Analyses
University of Birzeit, Palestine, 2015

Online Course

Watch Do ad Slides

Mustafa Jarrar
Birzeit University, Palestine
mjarrar@birzeit.edu

Watch this lecture and download the slides from

http://jarrar-courses.blogspot.com/2011/11/artificial-intelligence-fall-2011.html

This lecture

- Enumeration Method
- Inference rules
- Resolution
- Forward and backward Chaining

Information and material largely based on [1]

Lecture Keywords:

Logic, Propositional Logic, Inference Methods, Deduction, Reasoning, Enumeration Method, Inference rules, Resolution, refutation theorem-proving technique, Forward Chaining, Backward Chaining, Conjunctive Normal Form,Horn clauses, entailment, Logical Implication, Soundness, Completeness ،satisfiable, Unsatisfiable

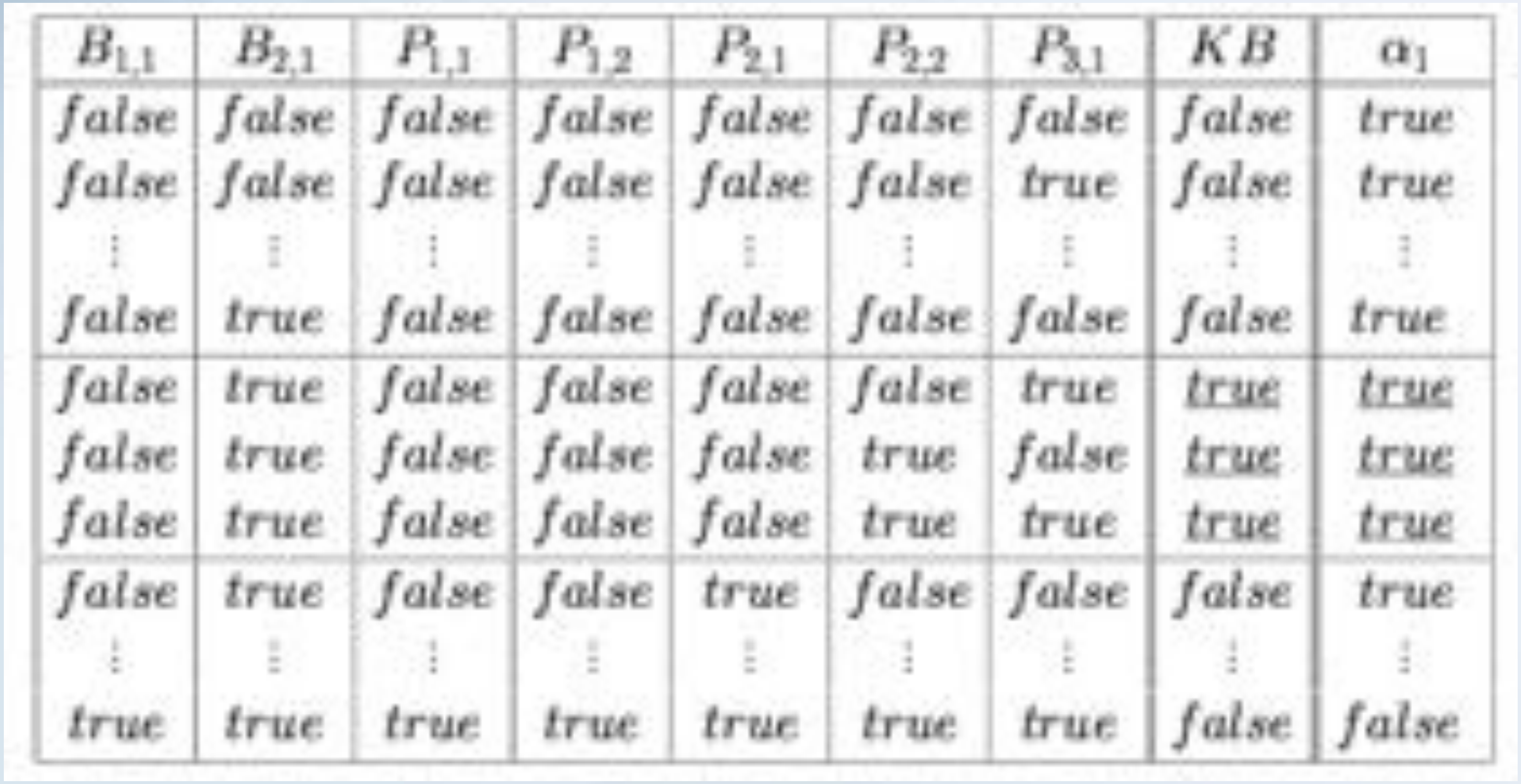
المنطق، المنطق الشكلي، الاستنتاج، الاستنباط، قواعد الاستنتاج،
طرق الاستنتاج ،صحة الجمل المنطقية، الحدود، التناقض

Inference Methods

- Enumeration Method
- Inference rules
- Resolution
- Forward and backward Chaining

Propositional Inference: Enumeration Method

Truth Tables for Inference

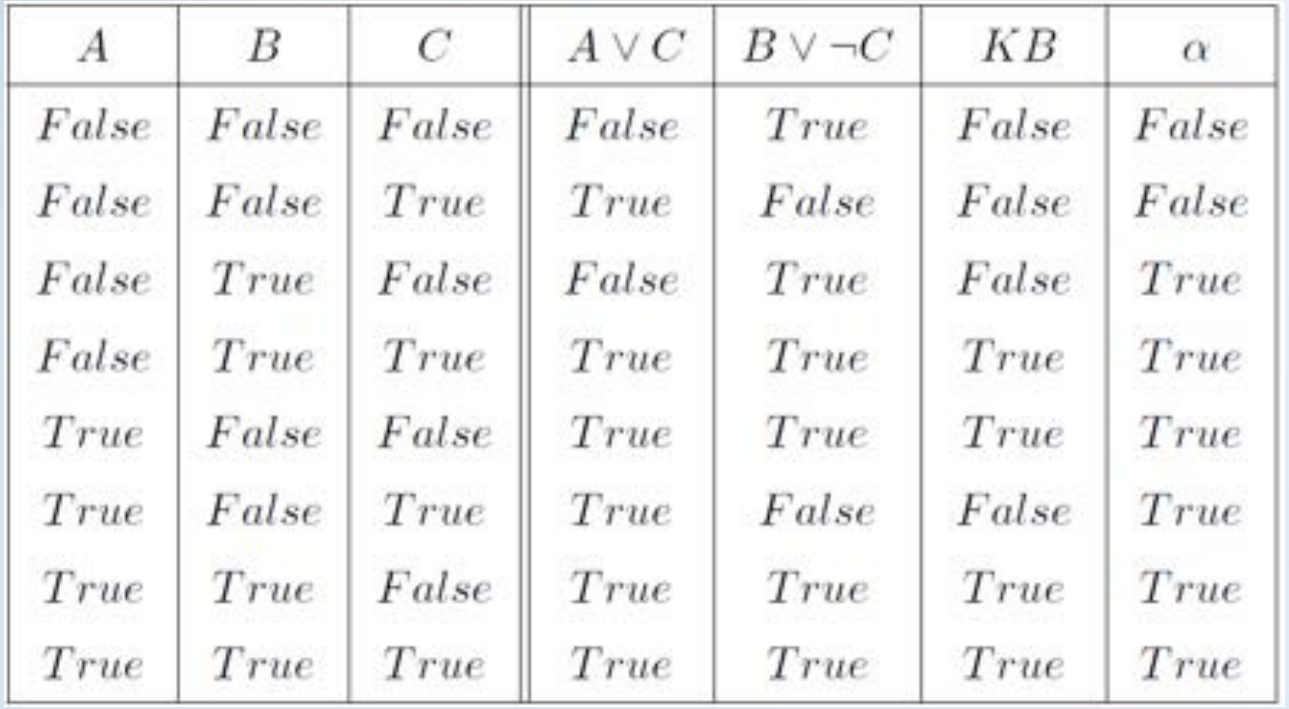


$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

Propositional Inference: Enumeration Method

- Let $\alpha = A \vee B$ and $KB = (A \vee C) \wedge (B \vee \neg C)$
- Is it the case that $KB \models \alpha$
- Check all possible models,

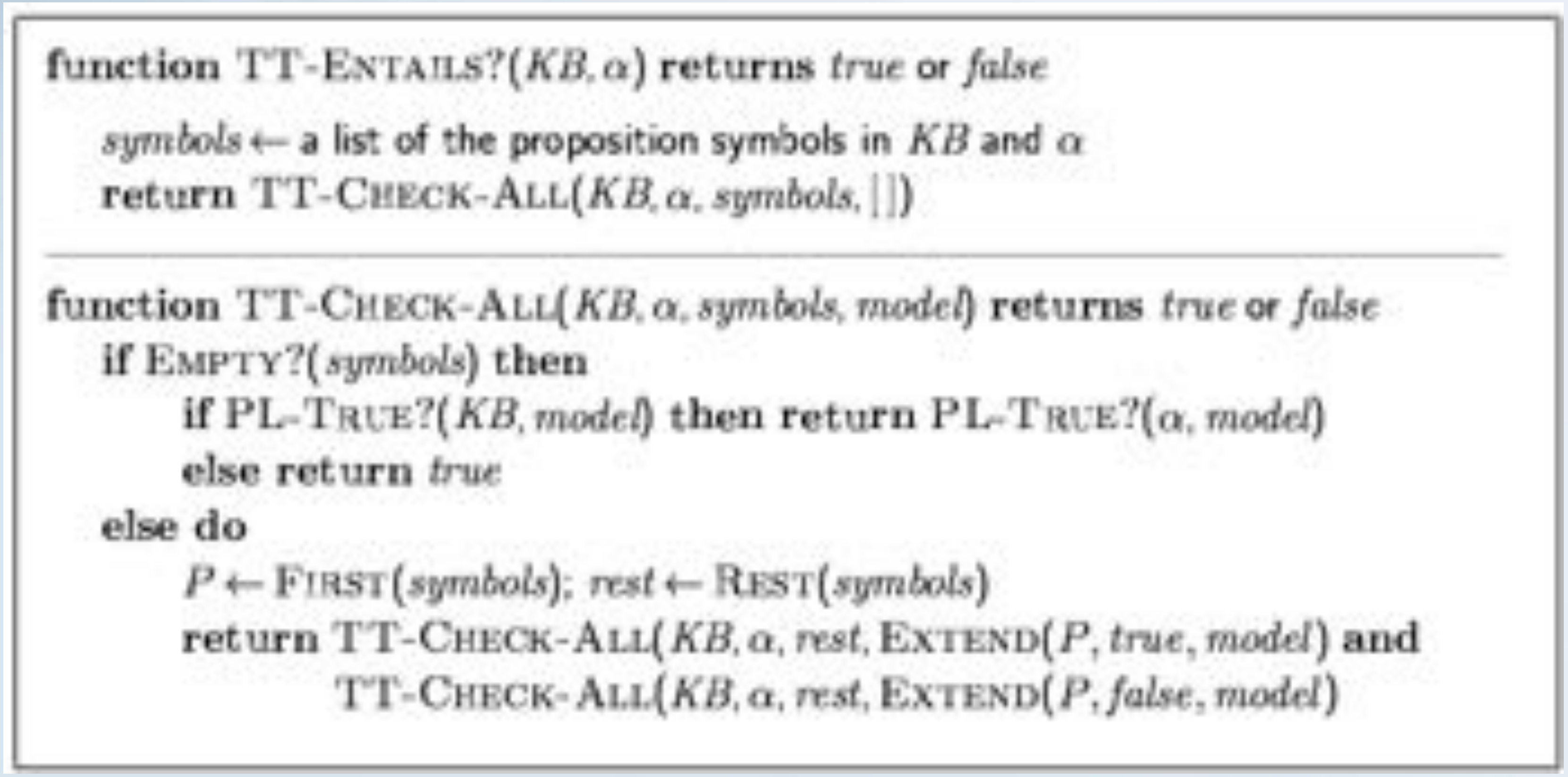
➔ α must be true wherever KB is true



A	B	C	$A \vee C$	$B \vee \neg C$	KB	α
$False$	$False$	$False$	$False$	$True$	$False$	$False$
$False$	$False$	$True$	$True$	$False$	$False$	$False$
$False$	$True$	$False$	$False$	$True$	$False$	$True$
$False$	$True$	$True$	$True$	$True$	$True$	$True$
$True$	$False$	$False$	$True$	$True$	$True$	$True$
$True$	$False$	$True$	$True$	$False$	$False$	$True$
$True$	$True$	$False$	$True$	$True$	$True$	$True$
$True$	$True$	$True$	$True$	$True$	$True$	$True$

Propositional Inference: Enumeration Method

- Depth-first enumeration of all models is sound and complete.
- For *n* symbols, time complexity is $O(2^n)$, space complexity is $O(n)$.



```
function TT-ENTAILS?(KB, α) returns true or false
    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
    if EMPTY?(symbols) then
        if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
        else return true
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
                    TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

Inference Methods

- Enumeration Method
- ➡ Inference rules
- Resolution
- Forward and backward Chaining

Inference Rule: Modus Ponens

The rule is written as: $$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

Means, whenever any sentences of the form $\alpha \Rightarrow \beta$ and α are given, then the sentence β can be inferred.

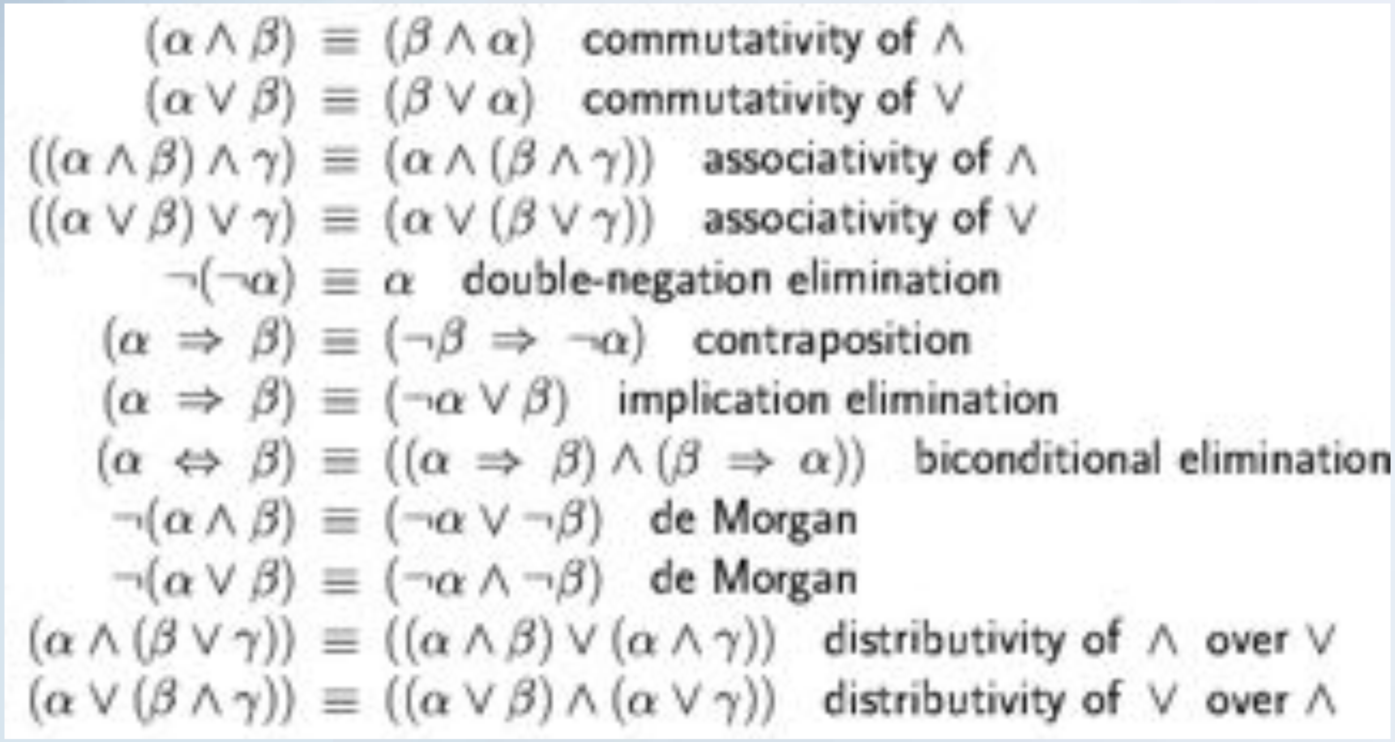
If α, then β.
α.

Therefore, β

For example, if (*WumpusAhead* $\wedge$ *WumpusAlive)* $\Rightarrow$ *Shoot* and (*WumpusAhead* $\wedge$ *WumpusAlive)* are given, then *Shoot* can be inferred.

More Inference Rules: Logical Equivalences

- Two sentences are logically equivalent iff true in same models: $\alpha \equiv$ ß *iff* $\alpha \models \beta$ and $\beta \models \alpha$



$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha) \equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	de Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	de Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

➔ All rules are sound if used with search algorithms, but they might be inadequate to reach a goal (i.e., completeness is not guaranteed).

Inference Methods

- Enumeration Method
- Inference rules
- Resolution

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

- Forward and backward Chaining

Resolution

Resolution is a rule of inference leading to a refutation (دحض) theorem-proving technique for sentences in propositional logic.

That is, applying the resolution rule in a suitable way allows for telling whether a propositional formula is satisfiable;

Resolution was introduced by John Alan Robinson in 1965.

Suppose we have a knowledge base in this form:

$$\frac{A \vee \cancel{B}, \qquad A \vee \cancel{\neg B}}{A}$$

By resolving ($A \vee B$) and ($A \vee \neg B$), we obtain ($A \vee A$), which is reduced to A

Notice that this rule applies only when a knowledge base in form of conjunctions of disjunctions of literals.

Resolution

We first write/convert the formulas into Conjunctive Normal Form (CNF):

conjunction of disjunctions of literals clauses

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

A literal is a propositional variable or the negation of a propositional variable.

- Resolution inference rule (for CNF):

$$\frac{l_i \vee \ldots \vee l_k, \qquad m_1 \vee \ldots \vee m_n}{l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

where l_i and m_j are complementary literals (one is the negation of the other).

E.g., $$\frac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}} \qquad \frac{a \vee b, \quad \neg a \vee c}{b \vee c}$$

➔ Resolution is **sound** and **complete** for propositional logic.

Conversion to CNF

Any sentence in propositional logic can be transformed into an equivalent sentence in Conjunctive Normal Form.

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

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

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

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

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

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

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

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

Resolution Algorithm

[2]

Any sentence in propositional logic can be transformed into an equivalent sentence in conjunctive normal form.

Steps:

- All sentences in KB and the *negation* of the sentence to be proved (the *conjecture*) are conjunctively connected.
- The resulting sentence is transformed into a conjunctive normal form with the conjuncts viewed as elements in a set, *S*, of clauses.
- The resolution rule is applied to all possible pairs of clauses that contain complementary literals. After each application of the resolution rule, the resulting sentence is simplified by removing repeated literals. If the sentence contains complementary literals, it is discarded (as a tautology). If not, and if it is not yet present in the clause set *S*, it is added to *S*, and is considered for further resolution inferences.
- If after applying a resolution rule the *empty clause* is derived, the complete formula is unsatisfiable (or *contradictory*), and hence it can be concluded that the initial conjecture follows from the axioms.
- If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, the conjecture is not a theorem of the original knowledge base.

Resolution Algorithm (in short)

- The resolution algorithm tries to prove:

$$KB \models \alpha \quad \text{equivalent to}$$
$$KB \wedge \neg\alpha \quad \text{unsatisfiable}$$

- Generate all new sentences from KB and the query.

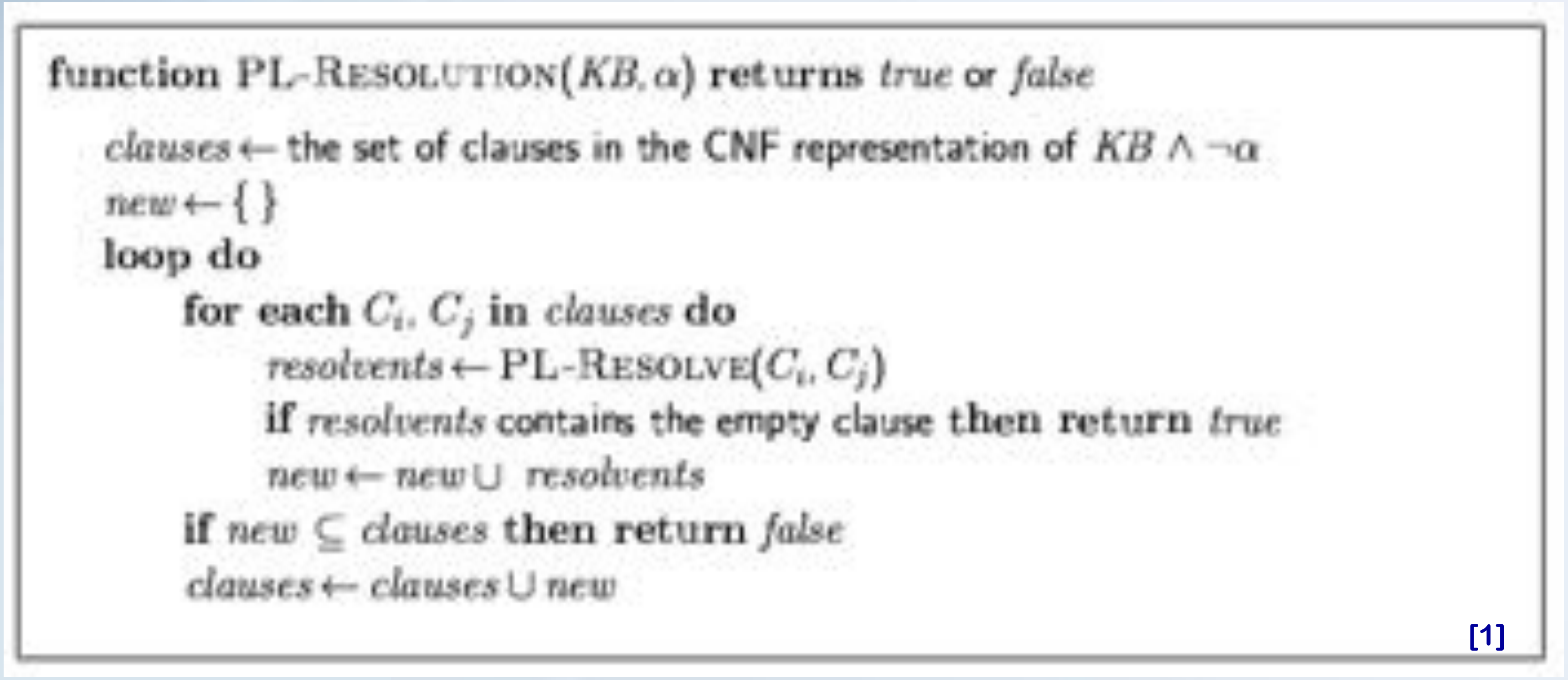
- One of two things can happen:

  1. We find a case like $P \wedge \neg P$ which is unsatisfiable, which means 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 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 $KB \wedge \neg\alpha$
    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
```

[1]

Example

KB = $(P \to Q) \to Q$

$(P \to P) \to R$

$(R \to S) \to \neg(S \to Q)$

α = R

Does KB entails α ($KB \models \alpha$)

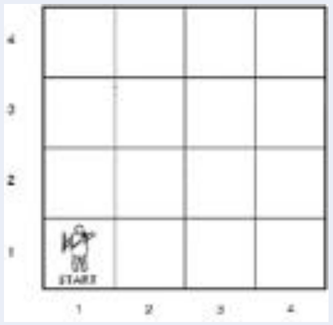
Contradiction!

1.	$P \vee Q$	
2.	$P \vee R$	
3.	$\neg P \vee R$	
4.	$R \vee S$	
5.	$R \vee \neg Q$	
6.	$\neg S \vee \neg Q$	
7.	$\neg R$	neg
8.	S	4,7
9.	$\neg Q$	6,8
10.	P	1,9
11.	R	3,10
12.	.	7,11

Exercise 1

KB = $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}))$ Breeze in [1,1] iff there is a Pit in [1,2] or [2.1].

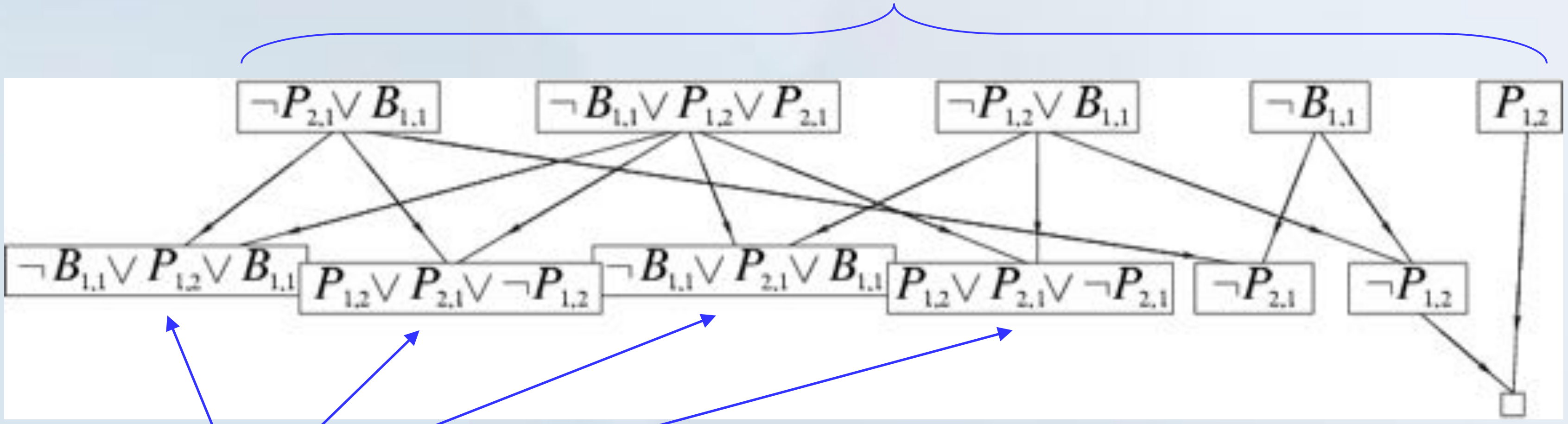
$\neg B_{1,1}$ There is on Breeze in [1,1]



$\alpha = \neg P_{1,2}$ No Pit in [1,2]?

Does KB entails α ($KB \models \alpha$)

$KB \wedge \neg\alpha$



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

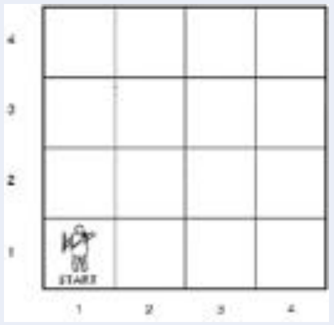
True!

False in all worlds

Exercise 2

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}))$ Breeze in [1,1] iff there is a it is [1,2] or [2.1].

$\neg B_{1,1}$ There is on Breeze in [1,1]



$\alpha = P_{1,2}$ Pit in [1,2]?

Does KB entails α ($KB \models \alpha$)

Completeness of the Resolution Method

- Self reading from the book
- You should be able to prove the completeness of the resolution method (at least informally).

Inference Methods

- Enumeration Method
- Inference rules
- Resolution
- Forward and backward Chaining

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 Horn clause has at **most** 1 positive literal.
  e.g. $A \vee \neg B \vee \neg C$
  $P1 \wedge P2 \wedge P3 \ldots \wedge Pn \rightarrow Q;$
  $\sim a \vee b \vee c \vee \sim d$ Not a Horn Clause

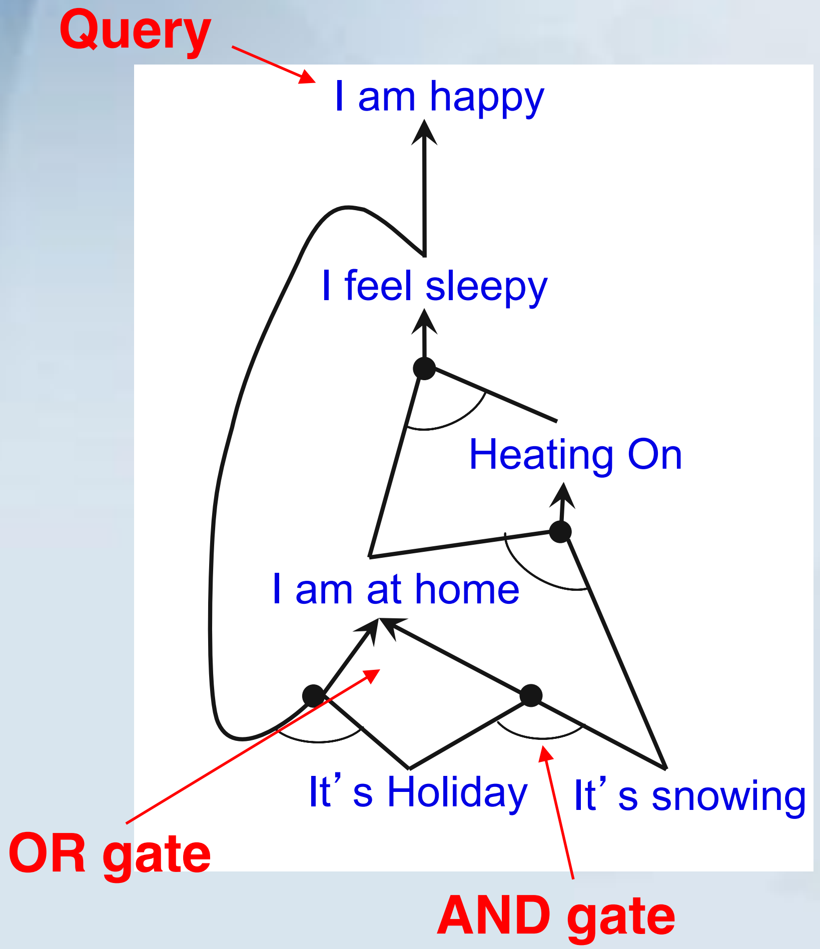
- Every Horn Clause can be rewritten as an implication with a conjunction of positive literals in the premises and a single positive literal as a conclusion. e.g. $B \wedge C \rightarrow A$

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

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

– add its conclusion to the *KB*, until query is found



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It' s snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It' s snowing ⇒ I am at home

Today is Holiday

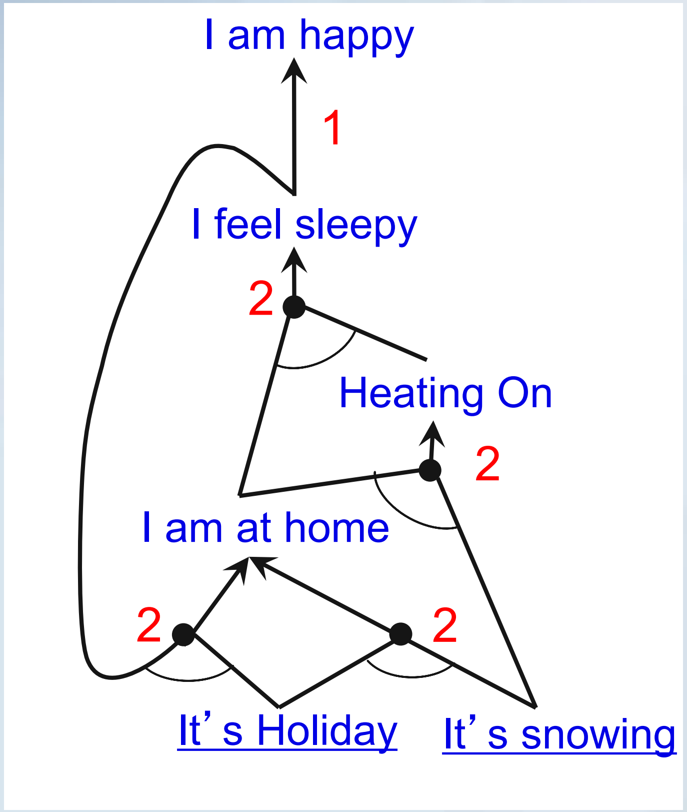
It' s snowing

➔ "I am happy"?

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

– add its conclusion to the *KB*, until query is found



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It' s snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It' s snowing ⇒ I am at home

Today is Holiday

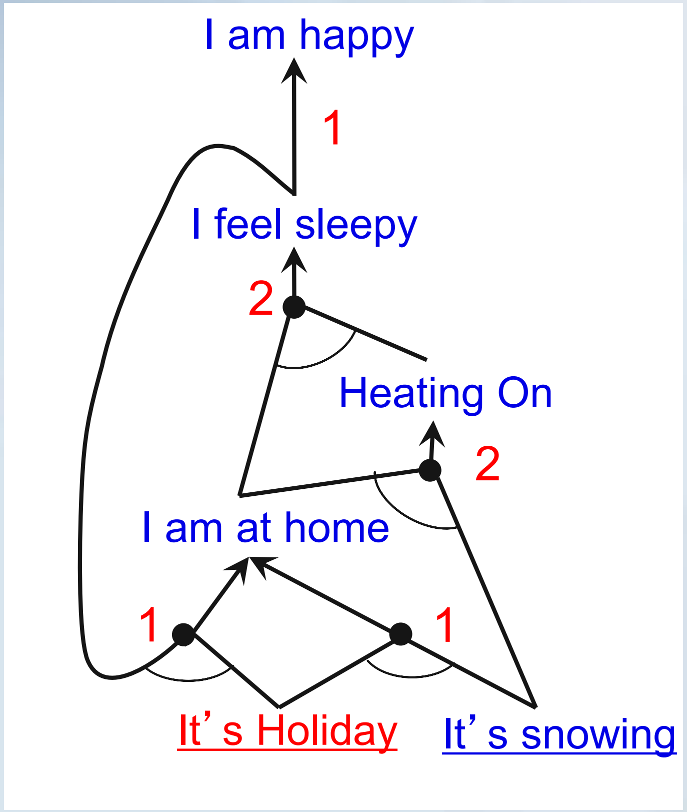
It' s snowing

➔ "I am happy"?

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

- add its conclusion to the *KB*, until query is found



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It’ s snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It’ s snowing ⇒ I am at home

Today is Holiday

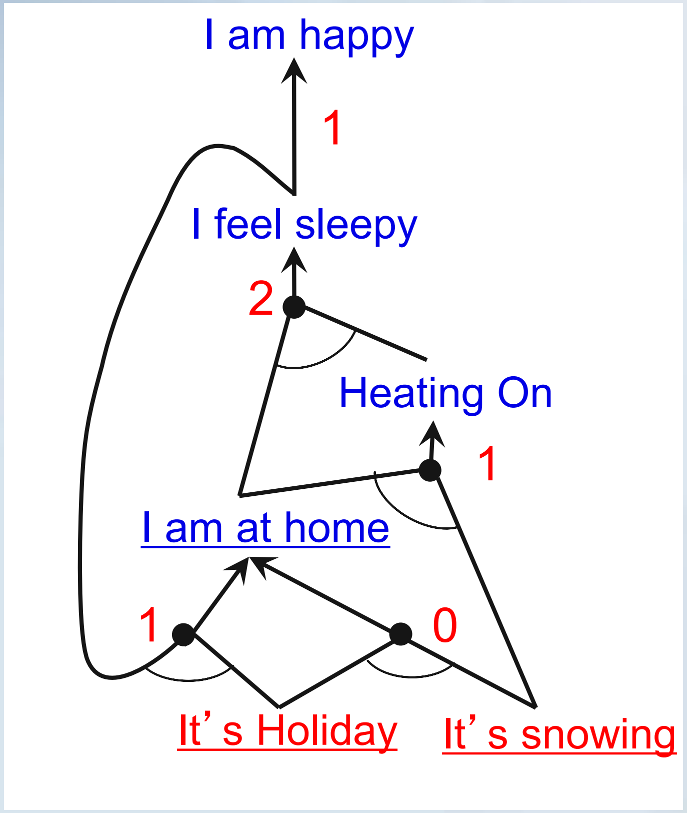
It’ s snowing

➔ “I am happy”?

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

– add its conclusion to the *KB*, until query is found



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It's snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It's snowing ⇒ I am at home

Today is Holiday

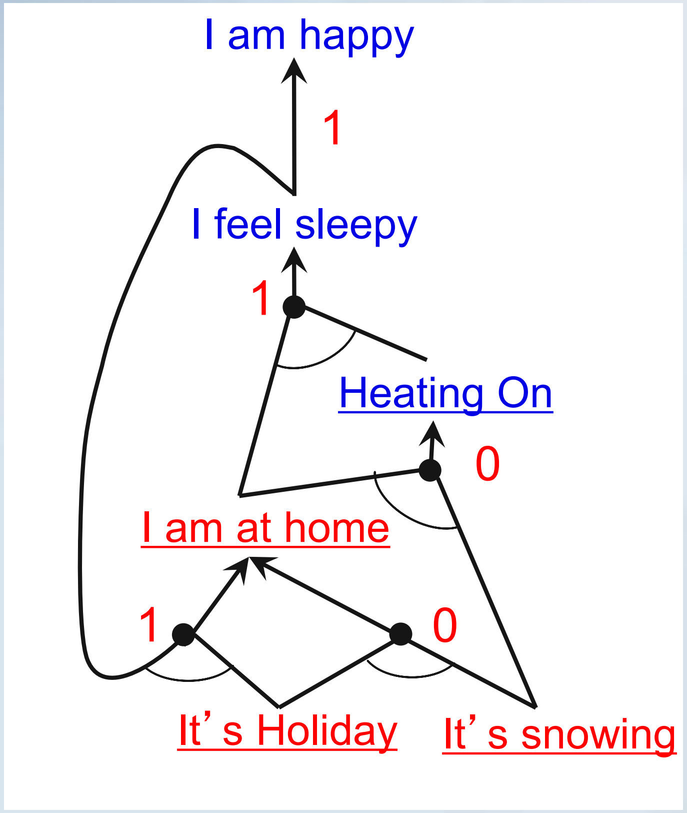
It's snowing

➔ "I am happy"?

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

- add its conclusion to the *KB*, until query is found



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It' s snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It' s snowing ⇒ I am at home

Today is Holiday

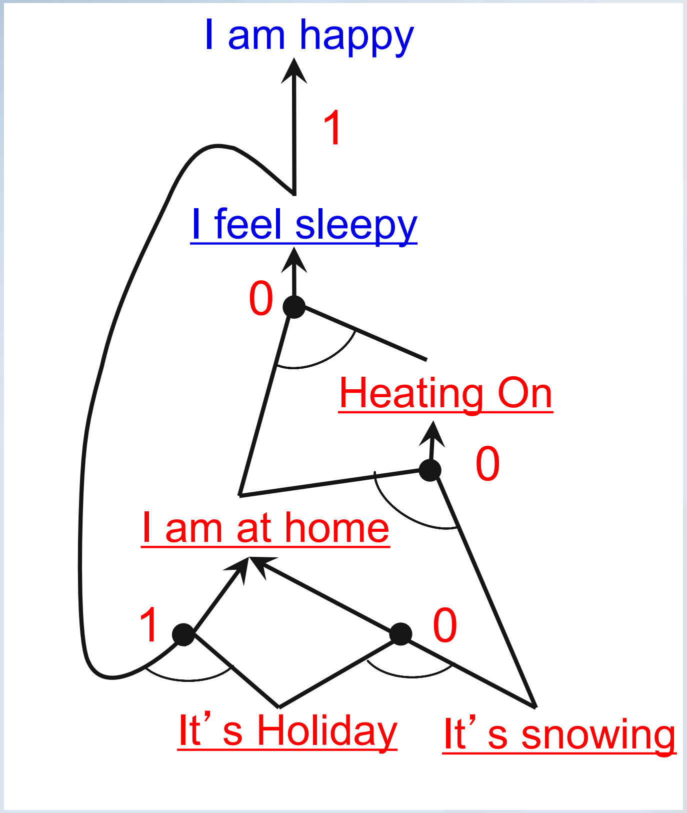
It' s snowing

➔ "I am happy"?

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

– add its conclusion to the *KB*, until query is found



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It' s snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It' s snowing ⇒ I am at home

Today is Holiday

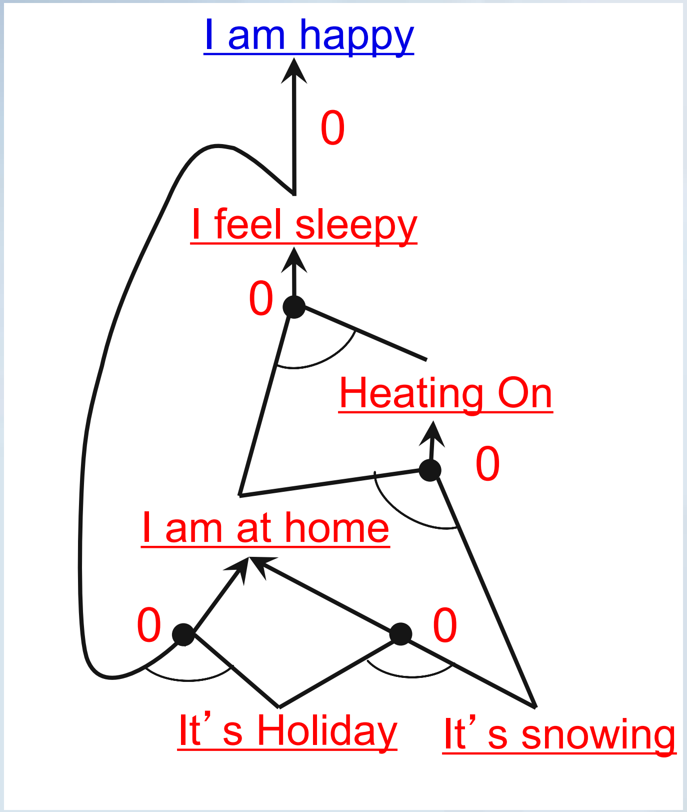
It' s snowing

➔ "I am happy"?

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

– add its conclusion to the *KB*, until query is found



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It' s snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It' s snowing ⇒ I am at home

Today is Holiday

It' s snowing

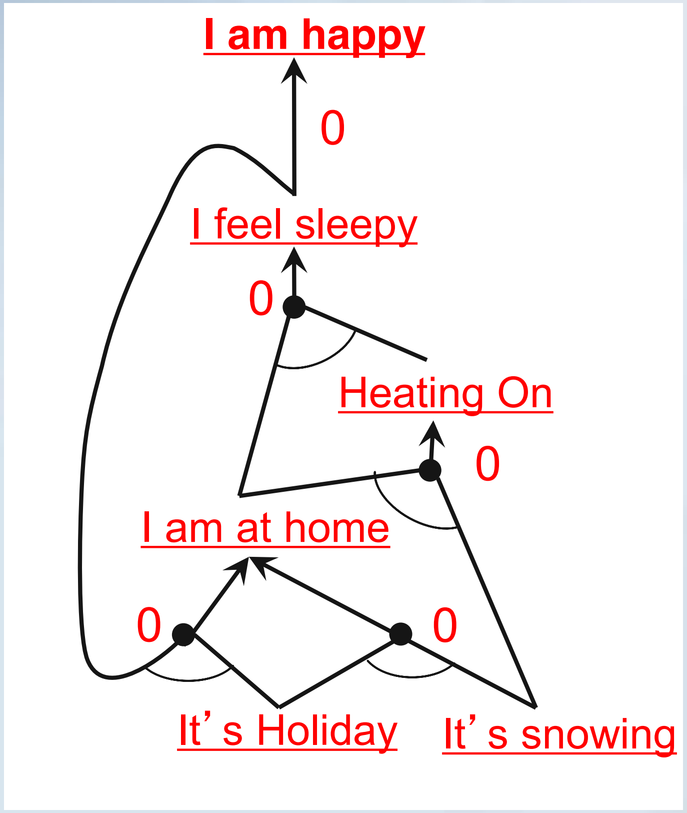
➔ "I am happy"?

Forward chaining example

Idea: fire any rule whose premises are satisfied in the *KB*,

– add its conclusion to the *KB*, until query is found

➔ Forward chaining is sound and complete for Horn KB



I feel sleepy ⇒ I am happy

I am at home ∧ heating On ⇒ I feel sleepy

I am at home ∧ It's snowing ⇒ Heating On

Today is Holiday ∧I feel sleepy ⇒ I am at home

Today is Holiday ∧ It's snowing ⇒ I am at home

Today is Holiday

It's snowing

➔ "I am happy"?

Think about this

Suppose that the goal is to conclude the color of a pet named Fritz, given that (**he croaks and eats flies)**, and that the Knowledge base contains the following :

1. **If** (X croaks and eats flies) - **Then** (X is a frog)
2. **If** (X chirps and sings) - **Then** (X is a canary)
3. **If** (X is a frog) - **Then** (X is green)
4. **If** (X is a canary) - **Then** (X is yellow)

Croaks	ينعق
Frog	ضفدع
Chirps	يغرد
Canary	كناري

This Knowledge base would be searched and the first rule would be selected, because its antecedent (If Fritz croaks and eats flies) matches our given data. Now the consequents (Then X is a frog) is added to the data. The rule base is again searched and this time the third rule is selected, because its antecedent (If Fritz is a frog) matches our data that was just confirmed. Now the new consequent (Then Fritz is green) is added to our data. Nothing more can be inferred from this information, but we have now accomplished our goal of determining the color of Fritz.

Backward Chaining

$$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$$

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.

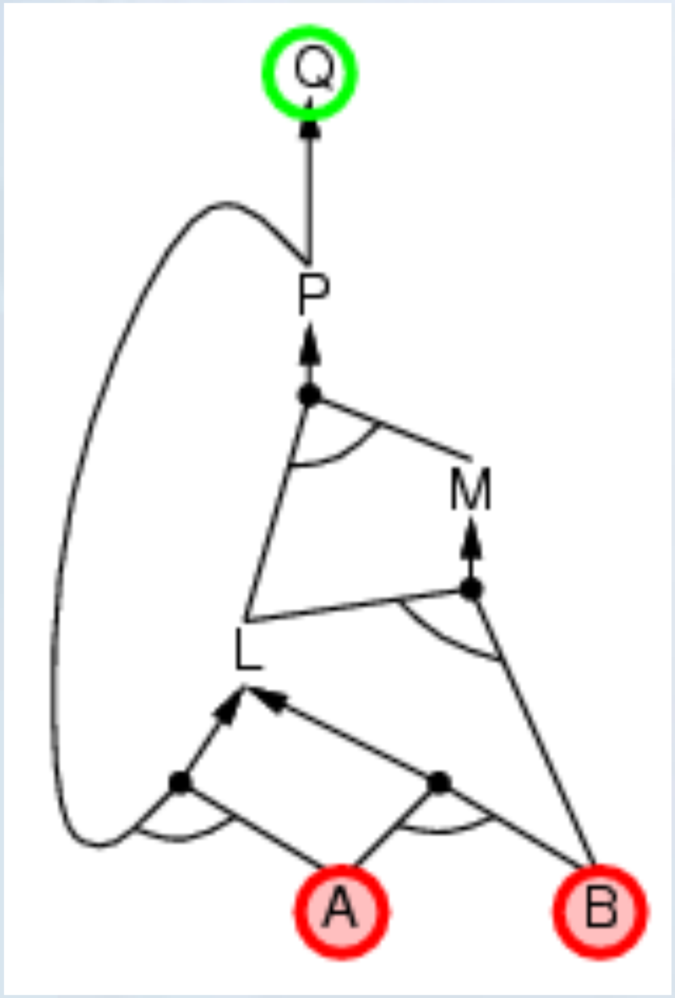
Avoid loops: check if new sub-goal is already on the goal stack

Avoid repeated work: check if new sub-goal

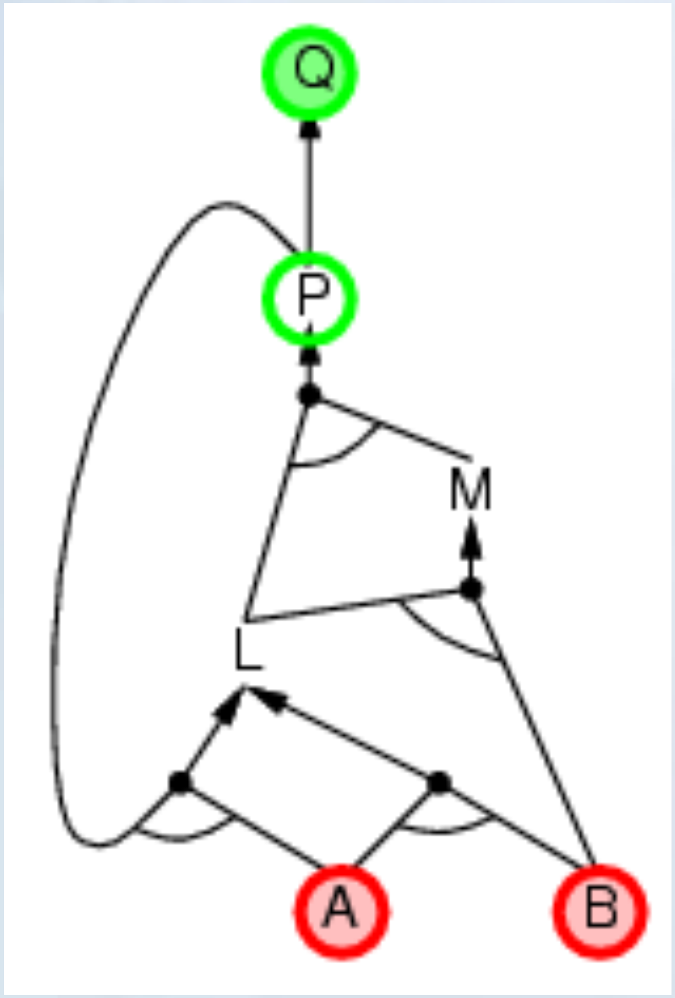
1. has already been proved true, or
2. has already failed

Backward chaining is the basis for "logic programming," e.g., Prolog

Backward chaining example

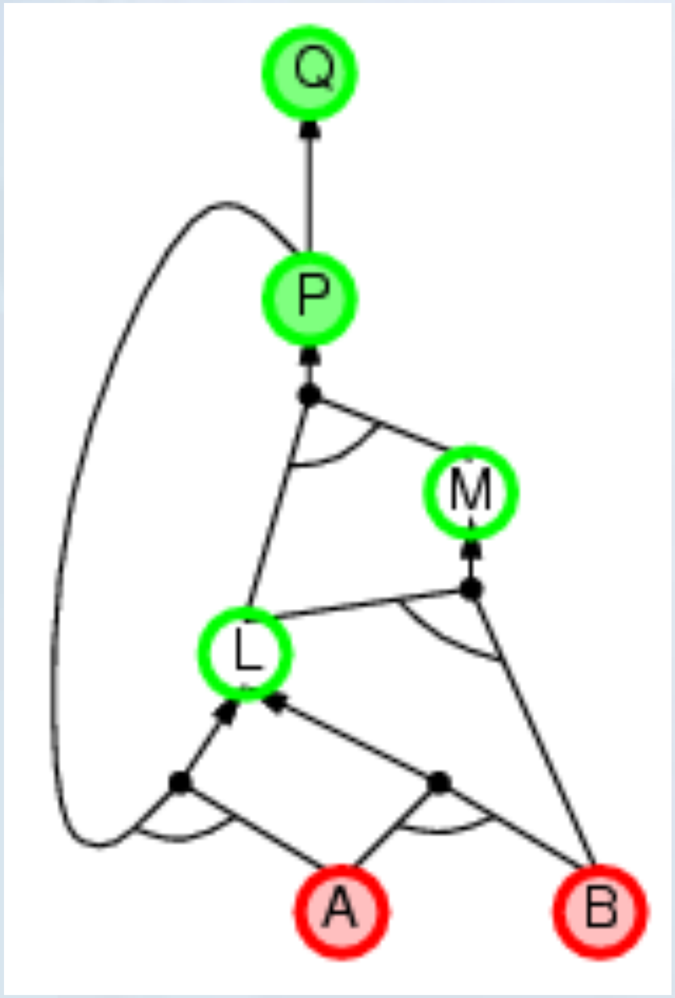


Backward chaining example



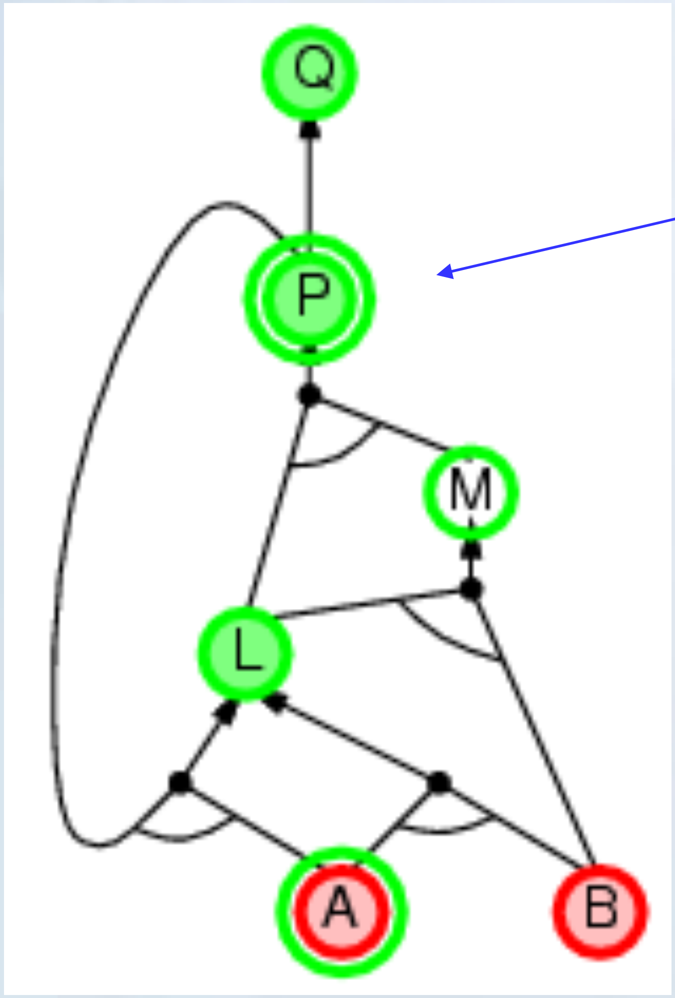
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

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

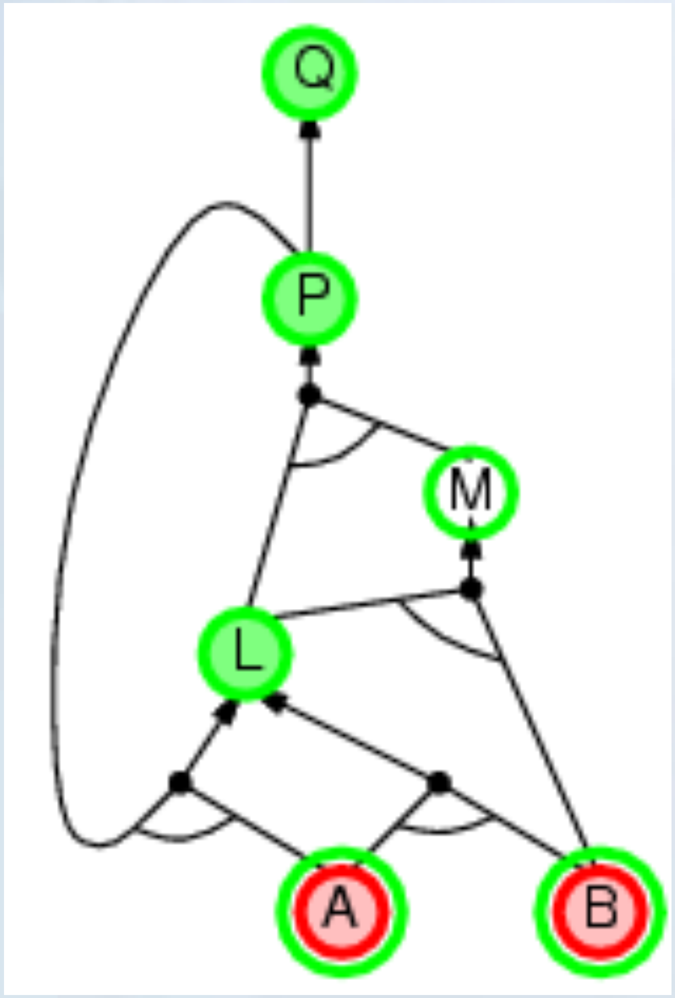
Backward chaining example



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

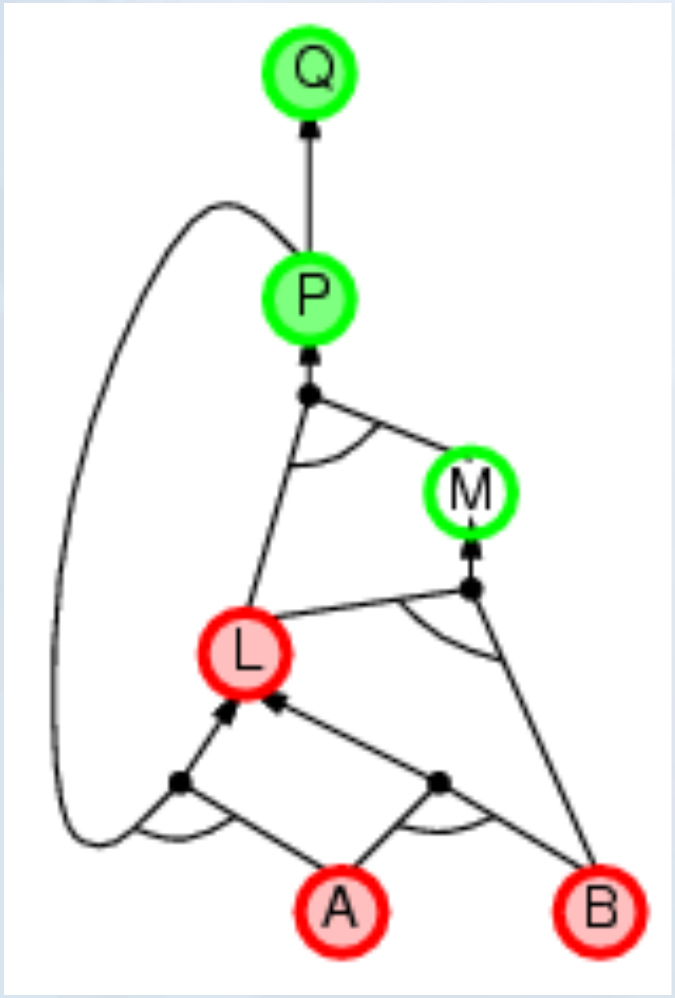
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward chaining example



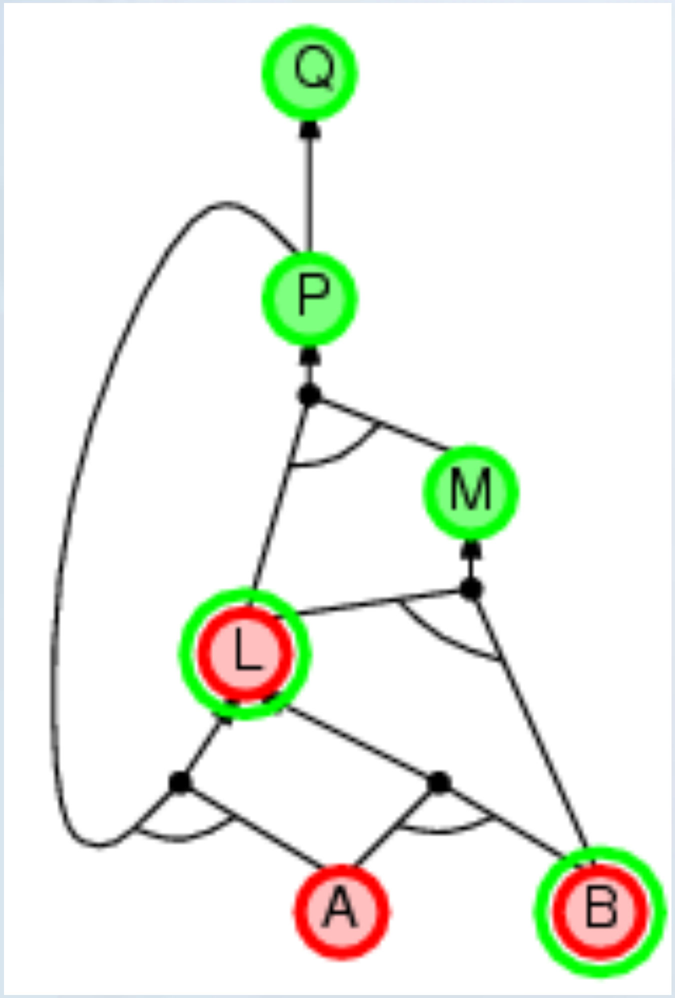
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward chaining example



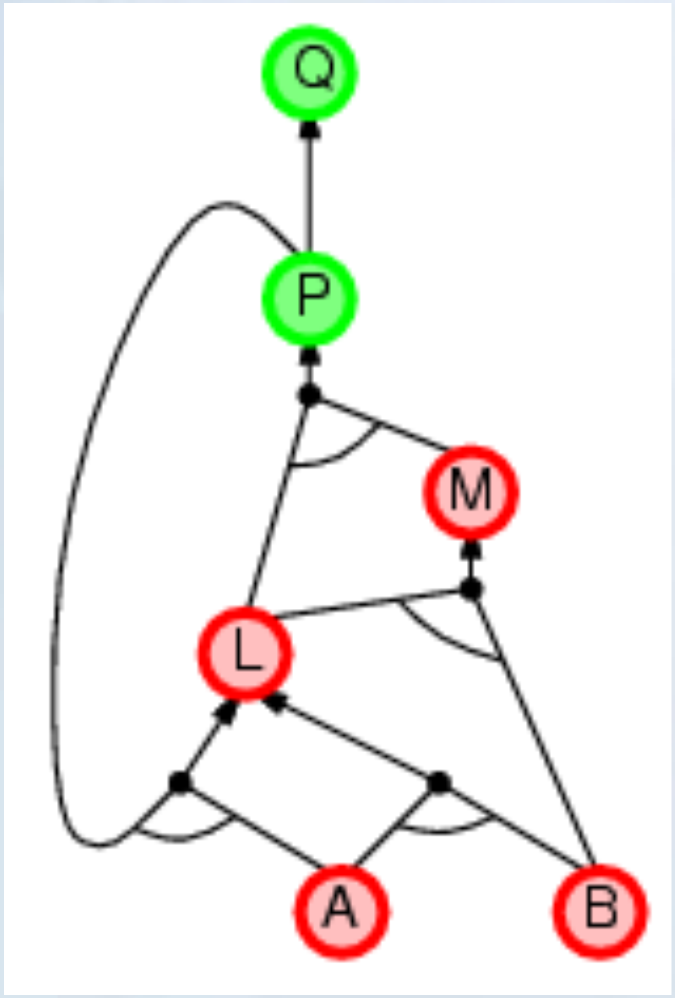
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

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



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

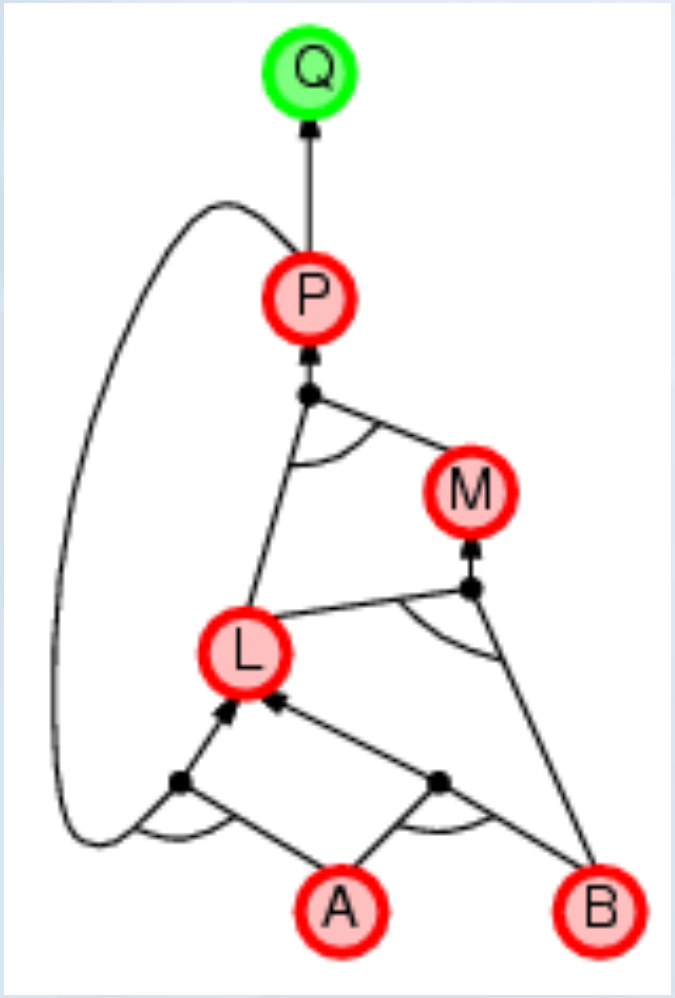
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

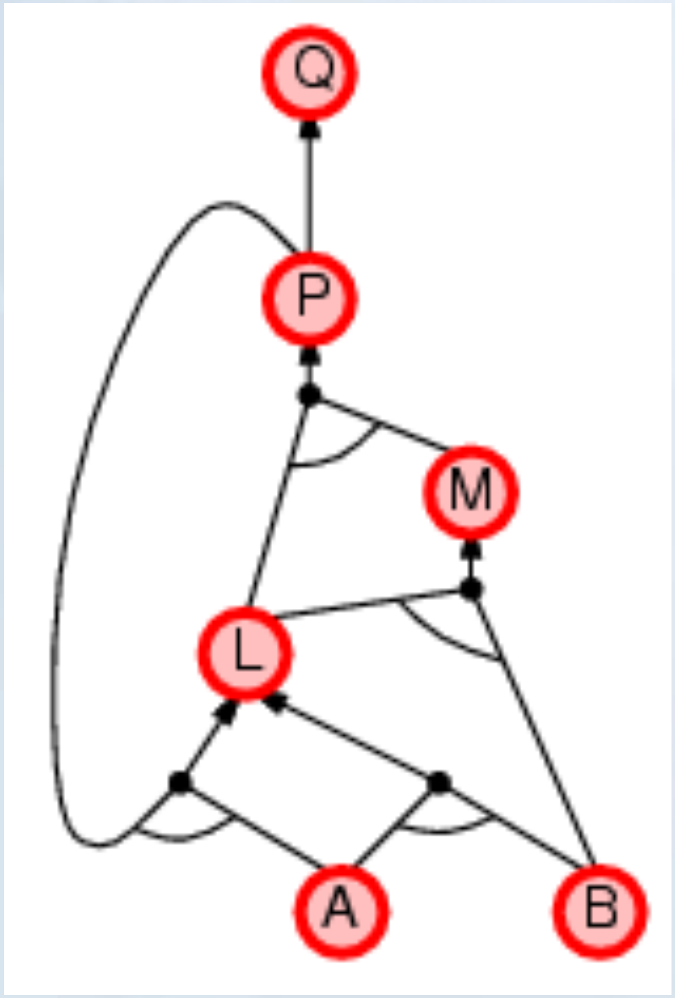
B

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, senseless processing,
  - e.g., object recognition, routine decisions
- May do lots of work that is irrelevant to the goal
- BC is goal-driven, (bottom-up reasoning) appropriate for problem-solving,
  - e.g., Where are my keys?
- Complexity of BC can be much less than linear in size of KB

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 sound and complete for propositional logic
- Forward, backward chaining are linear-time, complete for Horn clauses
- **Propositional logic lacks expressive power**

References

[1] S. Russell and P. Norvig: Artificial Intelligence: A Modern Approach Prentice Hall, 2003, Second Edition

[2] Leitsch, Alexander (1997), The resolution calculus, EATCS Monographs in Theoretical Computer Science, Springer, p. 11, Before applying the inference method itself, we transform the formulas to quantifier-free conjunctive normal form.