Chapter 07
Logic Agent

Instructor

LE Thanh Sach, Ph.D.

Instructor's Information

LE Thanh Sach, Ph.D.

Office:

Department of Computer Science,

Faculty of Computer Science and Engineering,

HoChiMinh City University of Technology.

Office Address:

268 LyThuongKiet Str., Dist. 10, HoChiMinh City, Vietnam.

E-mail: LTSACH@cse.hcmut.edu.vn

E-home: http://cse.hcmut.edu.vn/~ltsach/

Tel: (+84) 83-864-7256 (Ext: 5839)

Acknowledgment

The slides in this PPT file are composed using the materials supplied by

- **Prof. Stuart Russell and Peter Norvig**: They are currently from University of California, Berkeley. They are also the author of the book "Artificial Intelligence: A Modern Approach", which is used as the textbook for the course

- **Prof. Tom Lenaerts**, from Université Libre de Bruxelles

Outline

- Logic - Introduction
- Knowledge-based agents
- Wumpus world
- Logic in general - models and entailment
- Propositional (Boolean) logic
- Equivalence, validity, satisfiability
- Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - resolution

Logic - Introduction

Sentences in natural language	Sentence in Logic
Socrates is a man	man(socrates)
Plato is a man	man(plato)
All men are mortal	$\forall X$ (man(X) → mortal(X))
Conclusions	
Socrates is mortal: mortal(Socrates)	
Plato is mortal: mortal(Plato)	

Logic - Introduction

Sentences
1. Marcus was a man. 2. Macus was a Pompeian. 3. All Pompians were Romans. 4. Caesar was a ruler. 5. All Romans were either loyal to Caesar or hated hime. 6. Everyone is loyal to someone. 7. People only try to assassinate rulers they are not loyal to. 8. Marcus tried to assassinate Caesar.
Conclusions
Marcus was not loyal to Caesar

Knowledge bases

- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - `Tell` it what it needs to know
- Then it can `Ask` itself what to do - answers should follow from the KB
- Agents can be viewed at the knowledge level
  i.e., what they know, regardless of how implemented
- Or at the implementation level
  - i.e., data structures in KB and algorithms that manipulate them

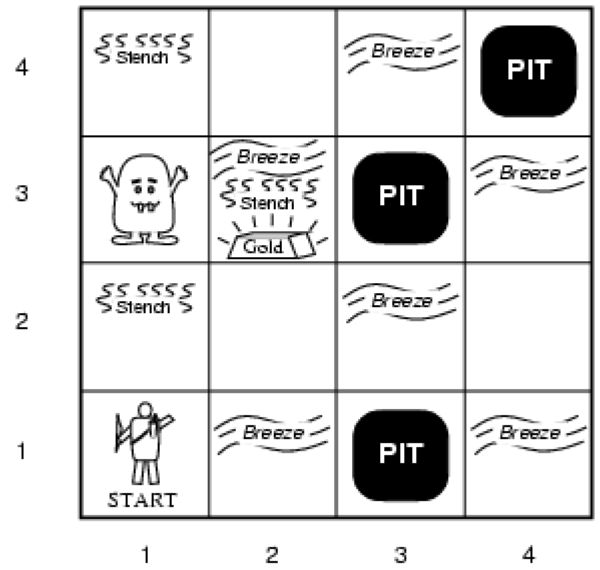
A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

- The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

Wumpus World PEAS description

- 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

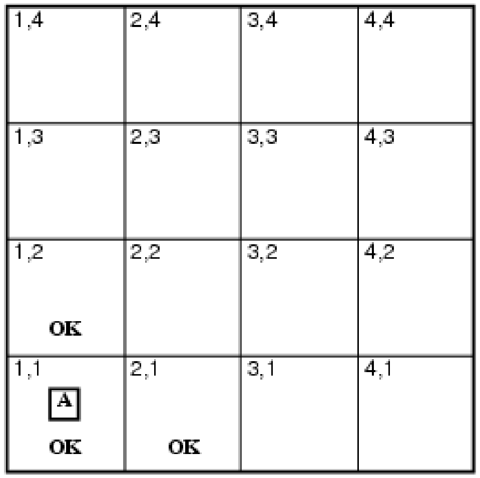


- Sensors: Stench, Breeze, Glitter, Bump, Scream
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot

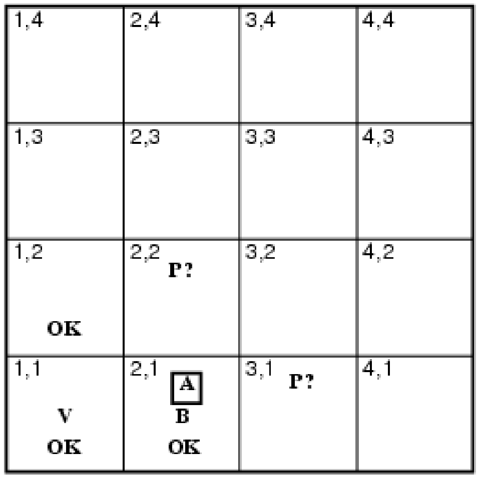
Wumpus world characterization

- Fully Observable No – only local perception
- Deterministic Yes – outcomes exactly specified
- Episodic No – sequential at the level of actions
- Static Yes – Wumpus and Pits do not move
- Discrete Yes
- Single-agent? Yes – Wumpus is essentially a natural feature

Exploring the Wumpus World



(a)

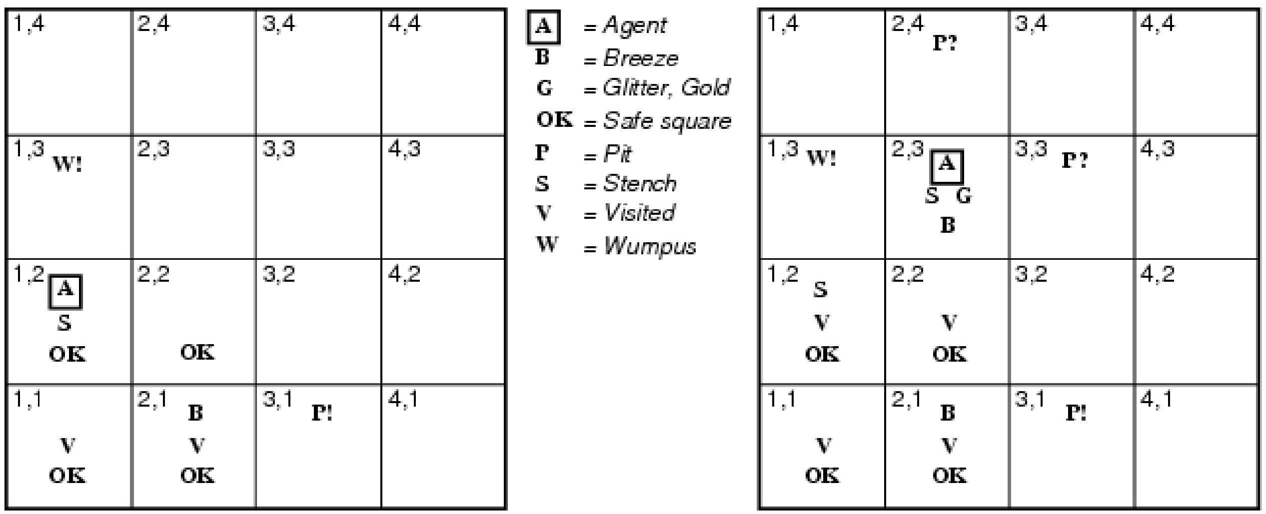


(b)

[1,1] The KB initially contains the rules of the environment. The first percept is [*none*, *none*,*none*,*none*,*none*], move to safe cell e.g. 2,1

[2,1] breeze which indicates that there is a pit in [2,2] or [3,1], return to [1,1] to try next safe cell

Exploring the Wumpus World



(a) (b)

[1,2] Stench in cell which means that wumpus is in [1,3] or [2,2]

YET … not in [1,1]

YET … not in [2,2] or stench would have been detected in [2,1]

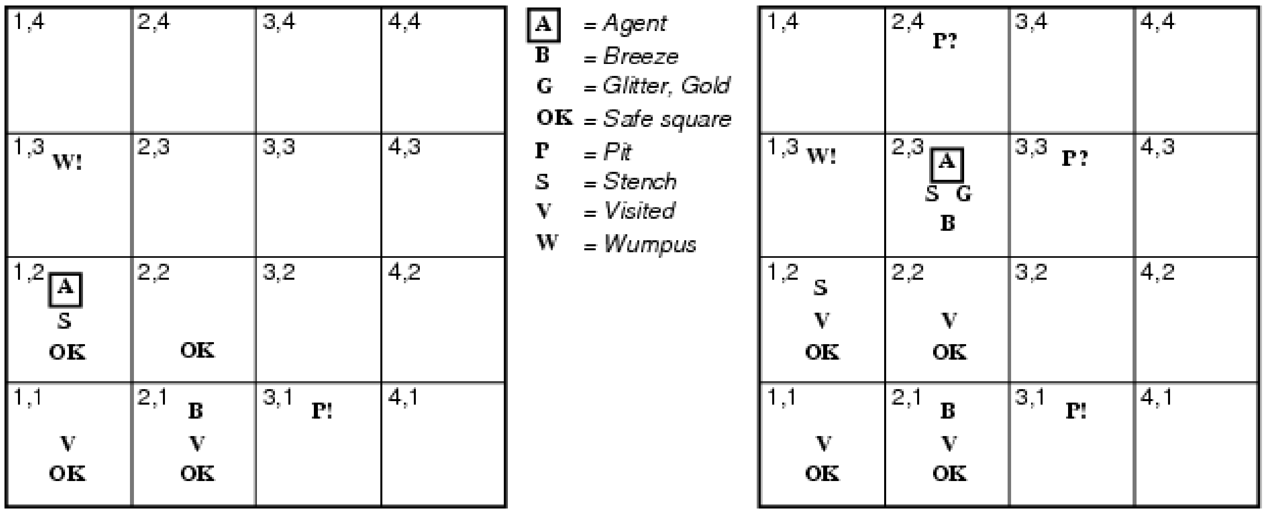
THUS … wumpus is in [1,3]

THUS [2,2] is safe because of lack of breeze in [1,2]

THUS pit in [1,3]

move to next safe cell [2,2]

Exploring the Wumpus World



(a)
(b)

[2,2] move to [2,3]
[2,3] detect glitter , smell, breeze
THUS pick up gold
THUS pit in [3,3] or [2,4]

Logic in general

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines the sentences in the language
- Semantics define the "meaning" of sentences;
  - i.e., define truth of a sentence in a world

- E.g., the language of arithmetic
  - $x+2 \geq y$ is a sentence; $x2+y >$ {} is not a sentence
  - $x+2 \geq y$ is true iff the number $x+2$ is no less than the number y
  - $x+2 \geq y$ is true in a world where $x = 7$, $y = 1$
  - $x+2 \geq y$ is false in a world where $x = 0$, $y = 6$

Entailment

- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- Knowledge base *KB* entails sentence α if and only if α is true in all worlds where *KB* is true
  - E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"
  - E.g., x+y = 4 entails 4 = x+y
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

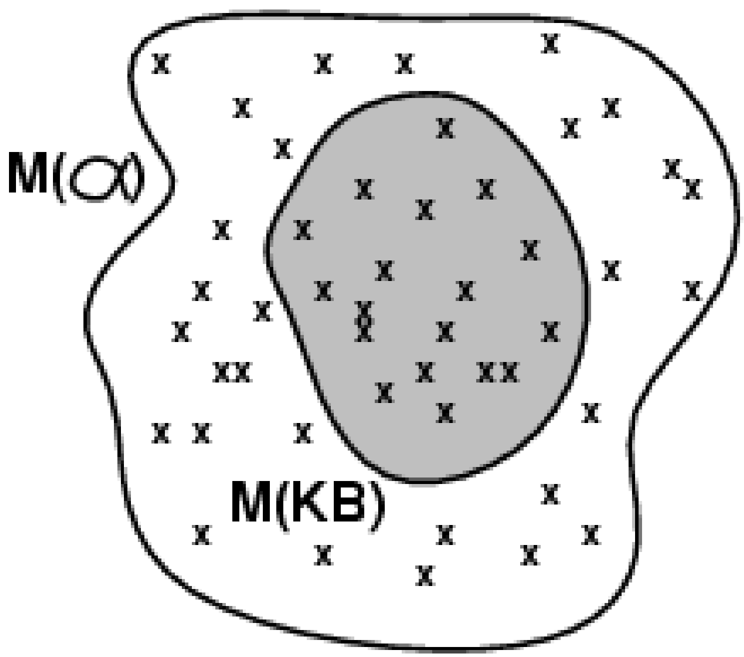
Models

- 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*

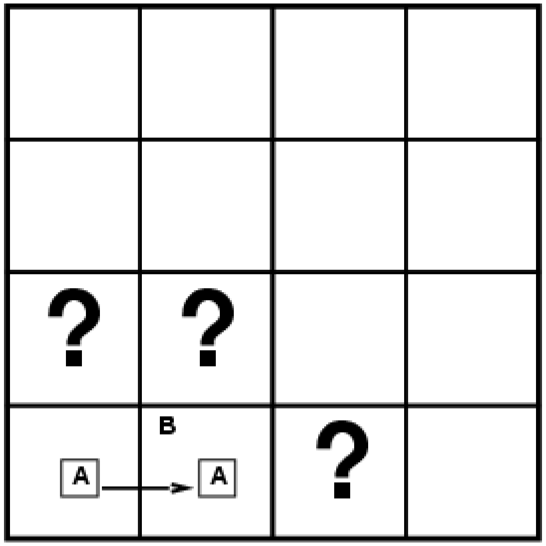
- *M(α)* is the set of all models of α

- Then KB ⊨ α iff $M(KB) \subseteq M(\alpha)$
  - E.g. *KB* = Giants won and Reds won α = Giants won



Entailment in the wumpus world

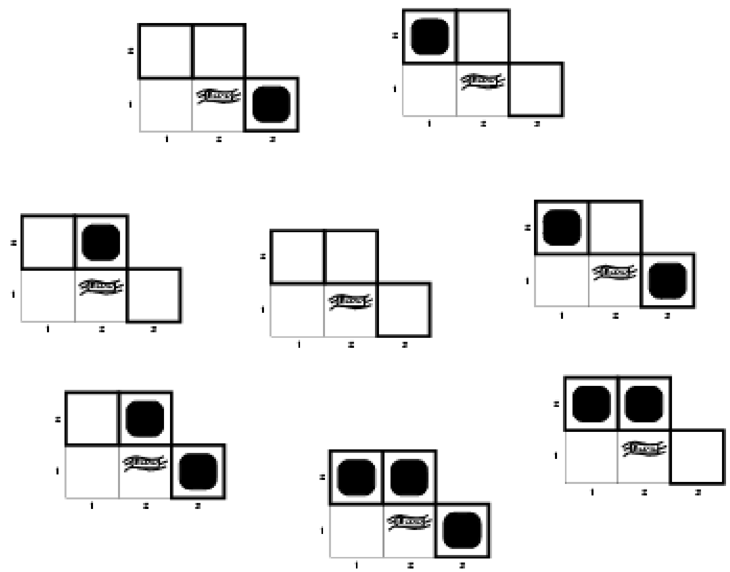
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]



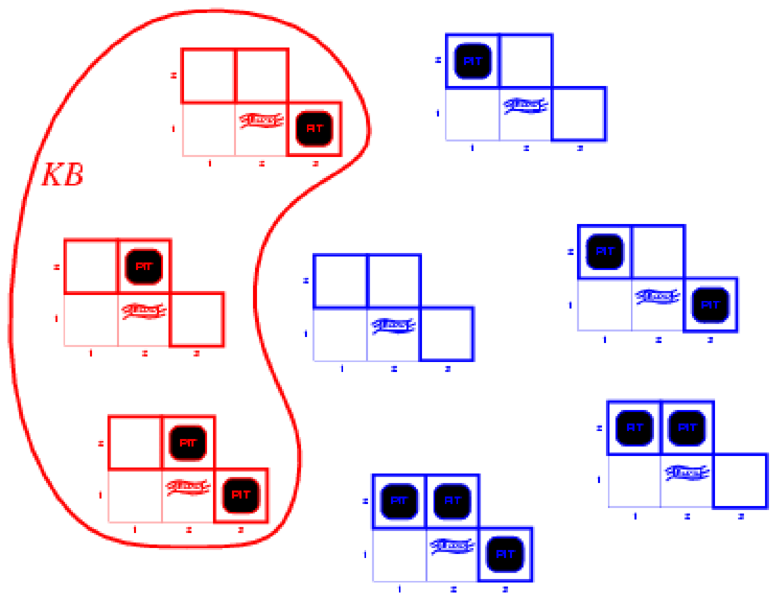
Consider possible models for *KB* assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models

Wumpus models

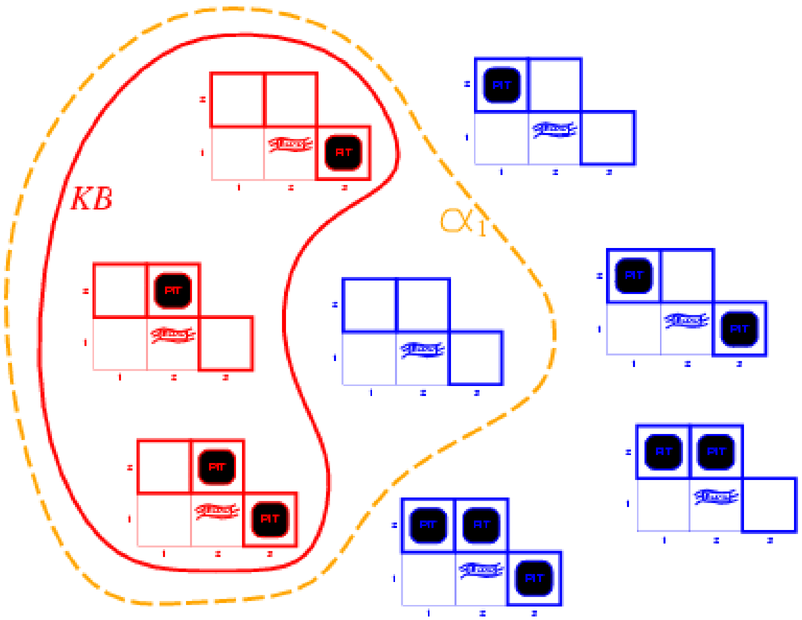


Wumpus models



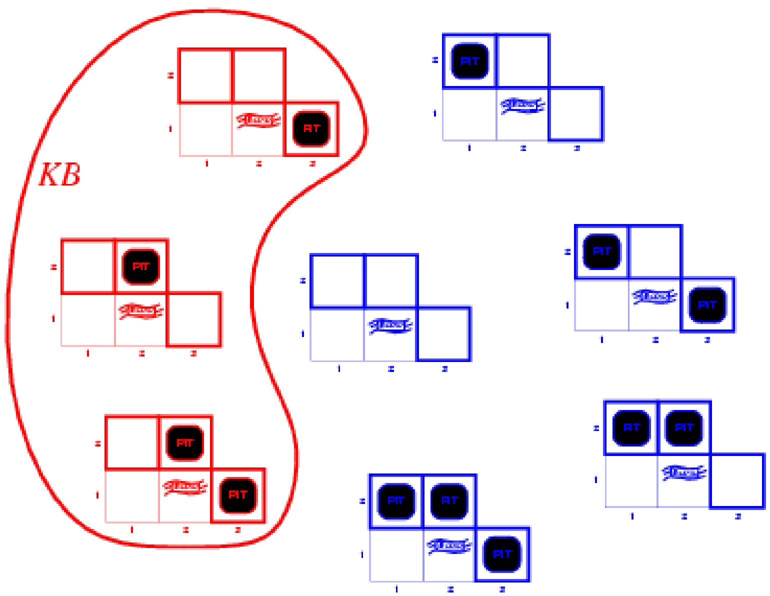
❖ *KB* = wumpus-world rules + observations

Wumpus models



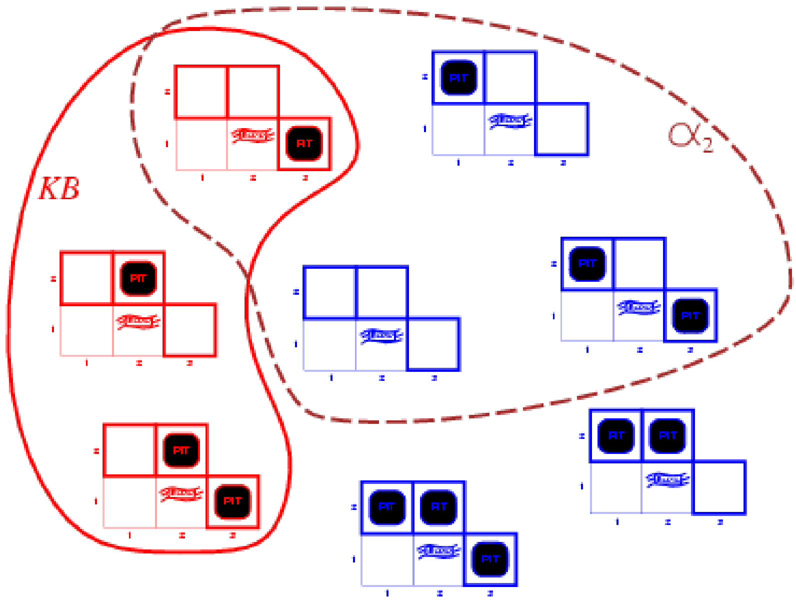
- *KB* = wumpus-world rules + observations
- α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by model checking

Wumpus models



❖ *KB* = wumpus-world rules + observations

Wumpus models



- KB = wumpus-world rules + observations
- α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

Inference

- *Notation:*
  - $KB \vdash_i \alpha$ is defined as "sentence α can be derived from *KB* by an algorithm *i*"
- Soundness:
  - If the algorithm only derives entailed sentences it is called *sound* or *truth preserving*.
  - *i* is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
- Completeness:
  - The algorithm can derive ANY sentence that is entailed.
  - *i* is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Inference

- 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*.

Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas

- The proposition symbols:
  - S, S_1, S_2 etc are sentences

  - If S is a sentence, $\neg S$ is a sentence (negation)
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	false	true	false

With these symbols, 8 possible models, can be enumerated automatically.

Rules for evaluating truth with respect to a model *m*:

$\neg S$	is true iff	S is false		
$S_1 \wedge S_2$	is true iff	S_1 is true	and	S_2 is true
$S_1 \vee S_2$	is true iff	S_1 is true	or	S_2 is true
$S_1 \Rightarrow S_2$	is true iff	S_1 is false	or	S_2 is true
i.e.,	is false iff	S_1 is true	and	S_2 is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true	and	$S_2 \Rightarrow S_1$ is true

Simple recursive process evaluates an arbitrary sentence, e.g.,

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (true \vee false) = true \wedge true = true$$

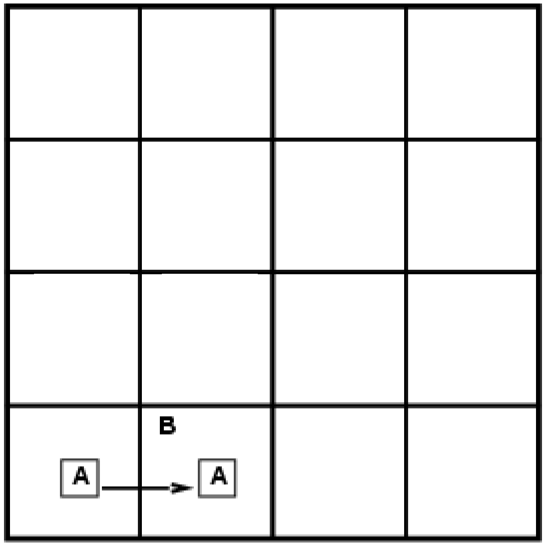
Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in [i, j].
Let $B_{i,j}$ be true if there is a breeze in [i, j].

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



❖ "Pits cause breezes in adjacent squares"

$$B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$$

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
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$false$	$true$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$false$	$true$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$	$false$

Inference by enumeration

❖

```
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))
```

❖ For *n* symbols, time complexity is $O(2^n)$, space complexity is $O(n)$

Logical equivalence

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

$$
\begin{aligned}
(\alpha \land \beta) &\equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) &\equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) &\equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) &\equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg\alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) &\equiv (\neg\alpha \lor \neg\beta) \quad \text{de Morgan} \\
\neg(\alpha \lor \beta) &\equiv (\neg\alpha \land \neg\beta) \quad \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) &\equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) &\equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{aligned}
$$

Validity and satisfiability

A sentence is valid if it is true in all models,

e.g., *True*, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in some model

e.g., $A \lor B$, C

A sentence is unsatisfiable if it is true in no models

e.g., $A \land \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable

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)
    - heuristic search in model space (sound but incomplete)
      e.g., min-conflicts-like hill-climbing algorithms

Resolution

- Conjunctive Normal Form (CNF)

  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):

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

where ℓ_i and m_j are complementary literals.

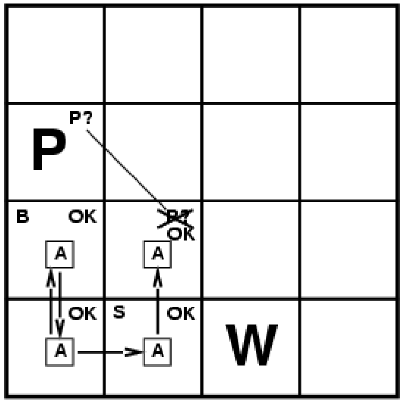
i.e., $\ell_i = \neg m_j$ or $m_j = \neg \ell_i$

Resolution

- Example:

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

- Resolution is sound and complete for propositional logic



Resolution

Soundness of resolution inference rule:

$$\frac{\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow l_i \qquad \neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}{\neg(l_i \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

Conversion to CNF

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

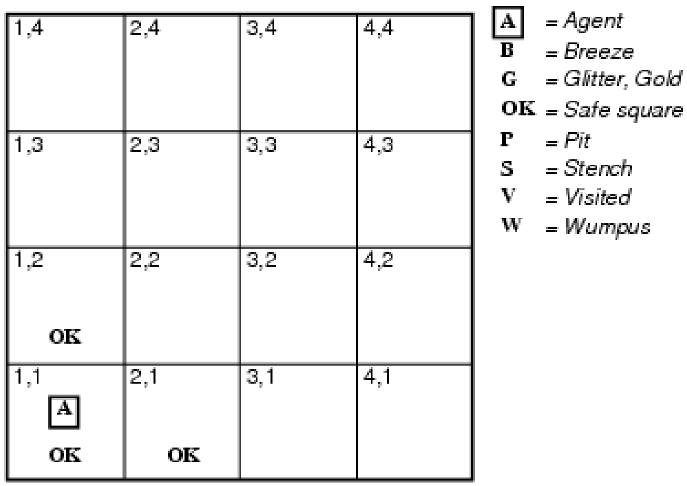
1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ 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:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution algorithm

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



(a)

- Game's rule:
  - [1,2] or [2,1] has Pit IFF [1,1] has Breeze
- Observation:
  - [1,1] has not Breeze
- Prove:
  - [1,2] has no Pit
  - [2,1] has not Pit

Resolution example

❖ Modeling the problem:

- $P_{1,2}$: [1,2] has Pit
- $P_{2,1}$: [2,1] has Pit
- $B_{1,1}$: [1,1] has Breeze

❖ Knowledge Base (KB):

1. $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$ ← [1,2] or [2,1] has Pit IFF [1,1] has Breeze
2. $\neg B_{1,1}$ ← [1,1] has no Breeze

Resolution example

- Prove:
  - [1,2] has no Pit
    i.e., $\neg P_{1,2}$
- Proof by contradiction
  - Add $\neg$ ($\neg P_{1,2}$) to KB to make a new KB
  - Prove the new KB to be unsatisfiable
  - = Find a contradiction in the new KB

Resolution example

- The new KB:
  1. $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  2. $\neg B_{1,1}$
  3. $\neg (\neg P_{1,2})$
- Convert the new KB to CNF
  1. Eliminate $\Leftrightarrow$ from $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

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

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

     Eliminate $\Rightarrow$ from $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$

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

Resolution example

- Convert the new KB to CNF
  1. Move $\neg$ inwards

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

     Apply distributivity law $\vee$ over $\wedge$

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

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

     Eliminate $\Rightarrow$ from $B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$

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

Resolution example

- Convert the new KB to CNF
  1. So, from $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ we has the following clauses:

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

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

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

Resolution example

- Convert the new KB to CNF

1. So, from $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ we has the following clauses:

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

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

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

2. $\neg B_{1,1}$
3. $P_{1,2}$

Resolution example

- Prove

#	Clauses	Note
1	$\neg P_{1,2} \vee B_{1,1}$	P
2	$\neg P_{2,1} \vee B_{1,1}$	P
3	$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$	P
4	$\neg B_{1,1}$	P
5	$P_{1,2}$	P
6	$B_{1,1}$	R:1,5
7	□	R: 4,6

P	= *premise*
R:i,j	= *Apply resolution rule to line (i) and (j)*
□	= contradiction

Resolution example

- *Exercises:*
  - Prove : $\neg P_{2,1}$

Resolution example

- *Exercises:*
  - Regarding the tsunami in Japan last month:
    - ✓ If Doremon's house is within 30Km from the nuclear plant in Fukushima then the ability that Doremon catchs radioactivity highly, so Doremon should move to a safety place. Doremon faces with difficulty when he moves to another place. Doremon needs supports from community and government if he faces with the diffculty.
  - If we know that
    - ✓ "Doremon's house is within 30Km from the nuclear plant in Fukushima"
  - Can we prove that
    - ✓ "Doremnon need supports from community and goverment"?

Resolution example

- *Mapping*
  - P_1 = "Doremon's house is within 30Km from the nuclear plant in Fukushima"
  - P_2 = "The ability that Doremon catchs radioactivity highly"
  - P_3 = "Doremon moves to a safety place"
  - P_4 = "Doremon faces with difficulty when he moves to another place"
  - P_5 = "Doremon needs supports from community and government"

Resolution example

- Mapping
  - $P_1 \Rightarrow P_2$
  - $P_2 \Rightarrow P_3$
  - $P_3 \Rightarrow P_4$
  - $P_4 \Rightarrow P_5$
  - P_1 : A fact
- Prove:
  - P_5

Resolution example

Example:

Given sentences:

1. P
2. (P ^ Q) → R
3. (S v T) → Q
4. T

Prove:

R

Resolution example

Expression	After standardization
P	P
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$
	$\neg T \vee Q$
T	T
$\neg R$	$\neg R$

Expression need to be proved: R, so add ($\neg$R)

Resolution example

#	Clauses	Note
1	P	Exiom (p)
2	¬P v ¬Q v R	p
3	¬S v Q	p
4	¬T v Q	p
5	T	p
6	¬R	P
7	¬P v ¬Q	2,6
8	¬Q	1,7
9	¬T	4,8
10	□	5,9

Prepositional Logic: Inference Rules

1. Modus ponens (MP):
   $A, A \rightarrow B \models B$

2. Modus Tollens (MT):
   $A \rightarrow B, \neg B \models \neg A$

3. Conjunction (Conj):
   $A, B \models A \wedge B$

4. Simplification (Simp):
   $A \wedge B \models A\ (B)$

Prepositional Logic: Inference Rules

5. Addition (Add):

   $A \models A \vee B$

6. Disjunctive syllogism (DS):

   $A \vee B, \neg A \models B$

7. Hypothetical syllogism (HS)

   $A \rightarrow B, B \rightarrow C \models A \rightarrow C$

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

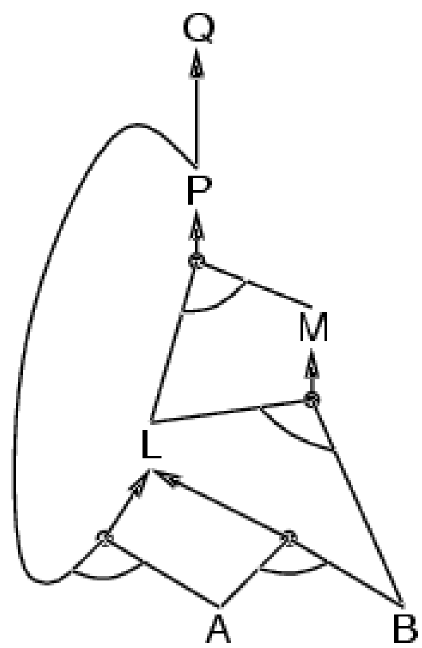
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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

Forward chaining

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

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



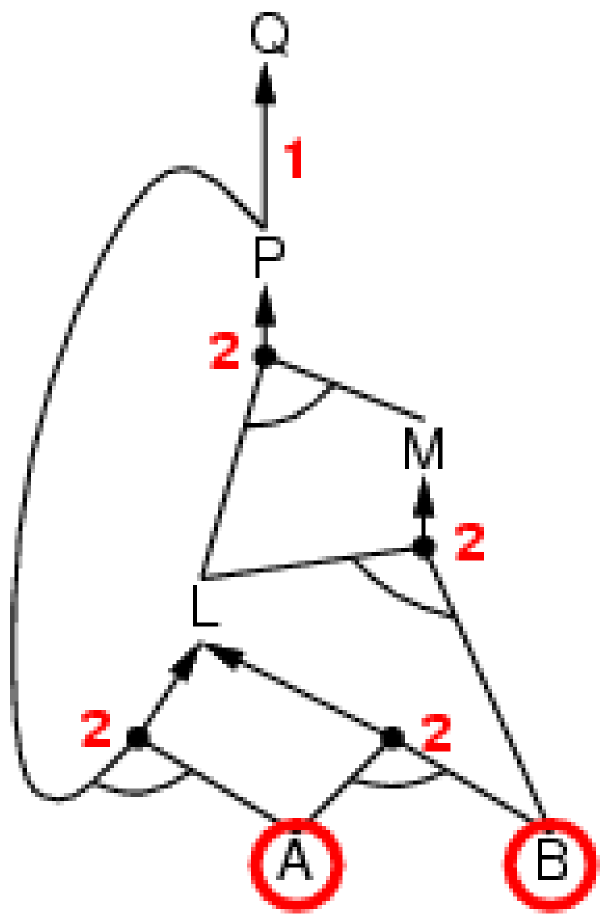
Prove: Q

Forward chaining algorithm

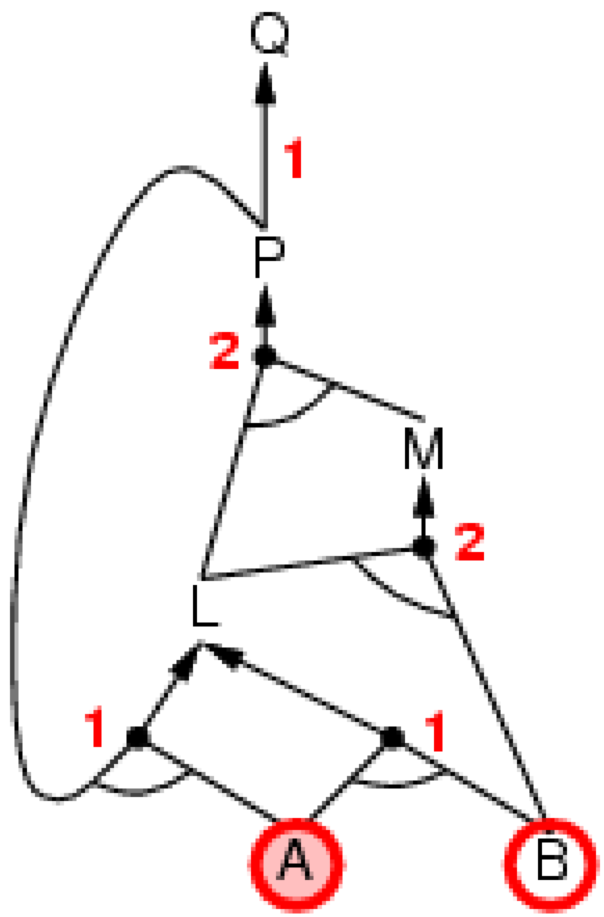
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

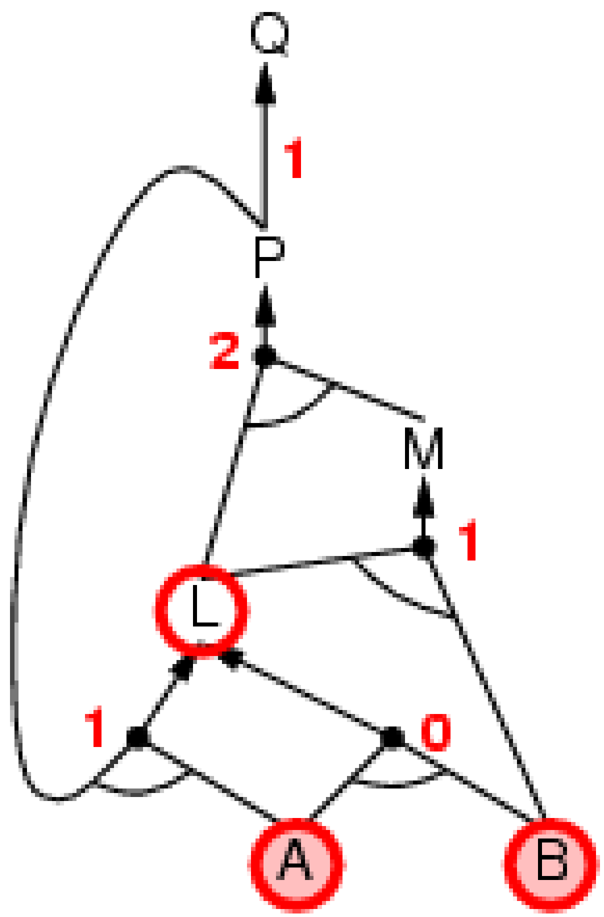
Forward chaining example



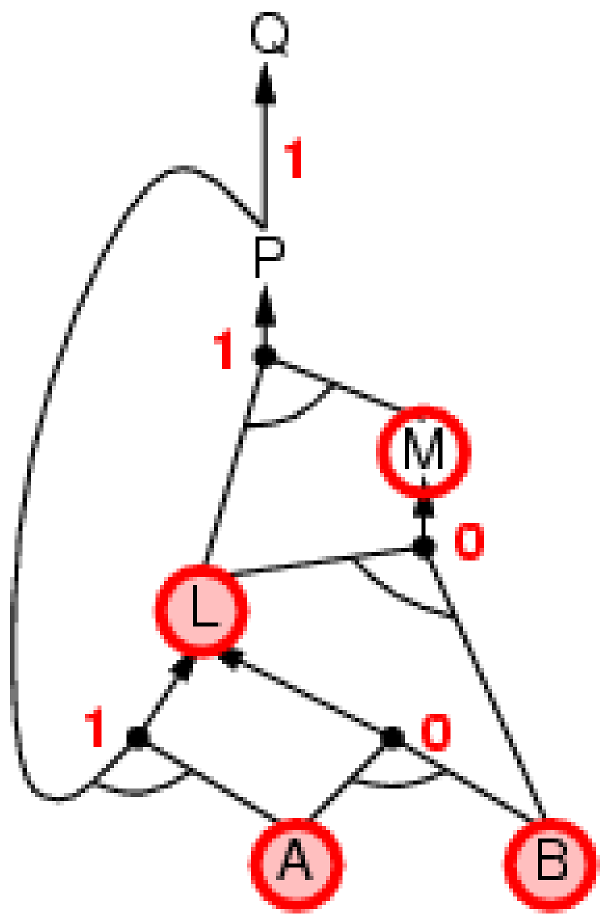
Forward chaining example



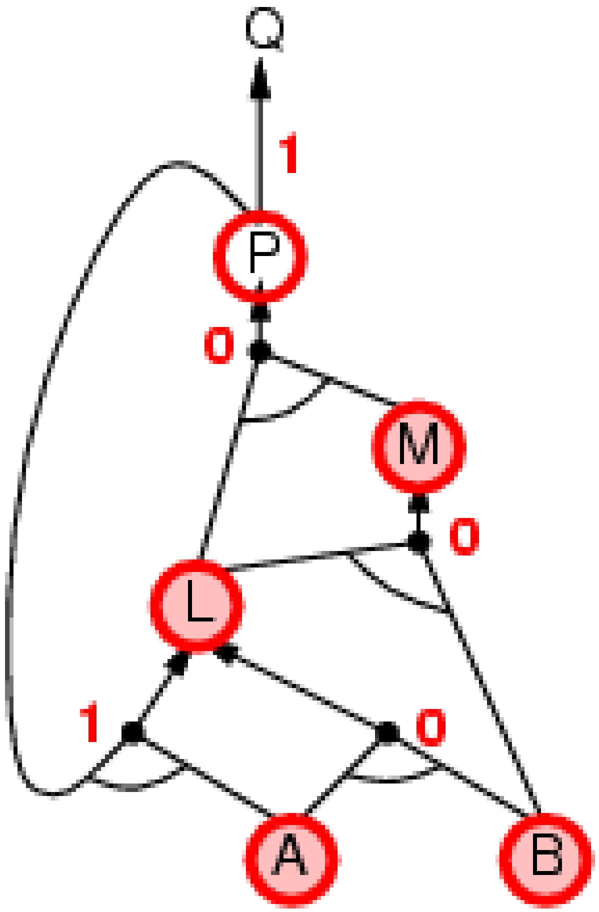
Forward chaining example



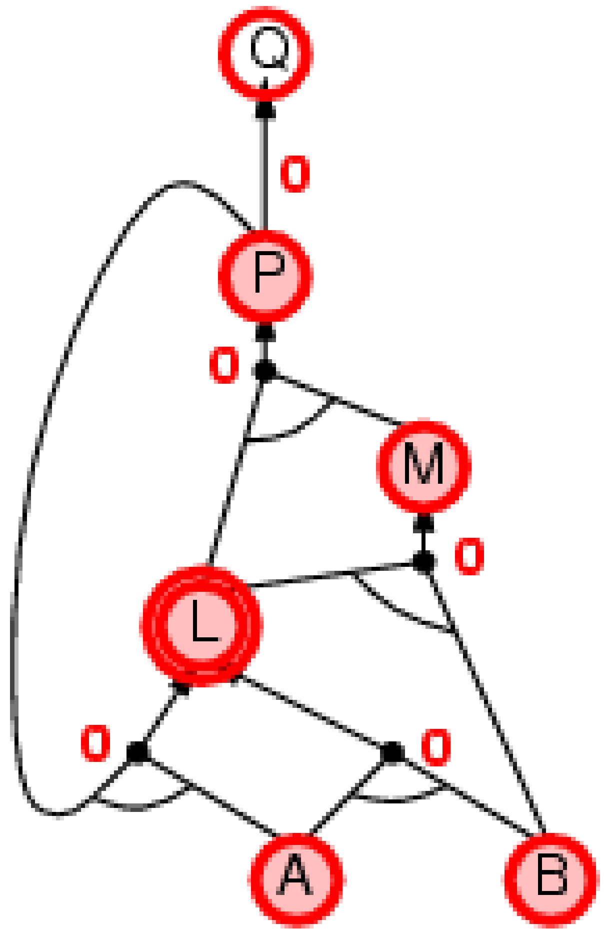
Forward chaining example



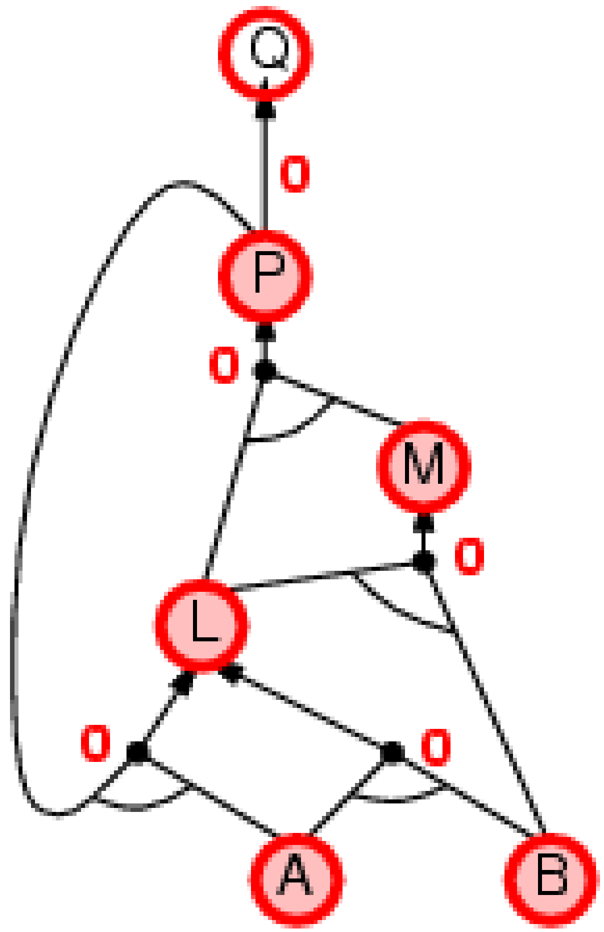
Forward chaining example



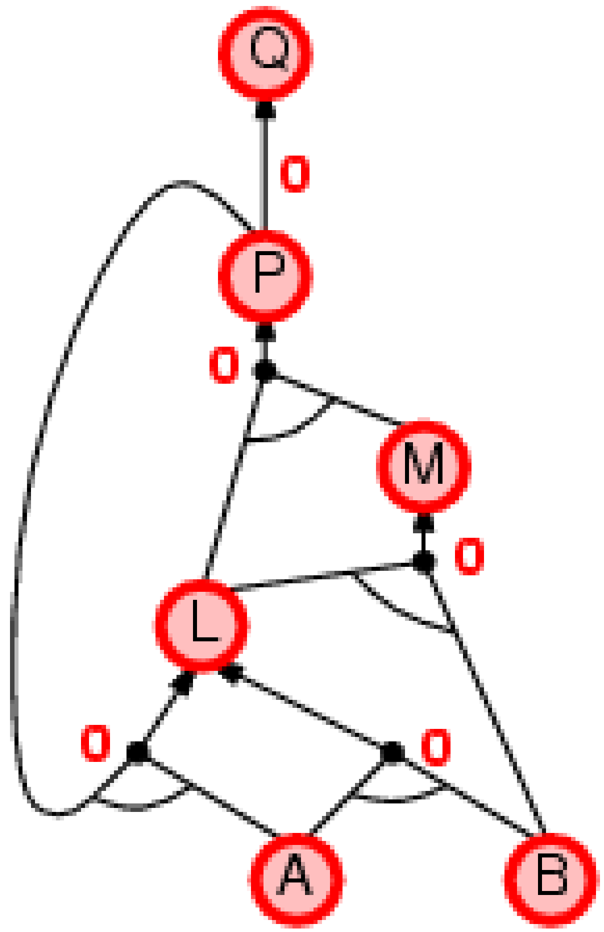
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

- FC derives every atomic sentence that is entailed by *KB*
  1. FC reaches a fixed point where no new atomic sentences are derived
  2. Consider the final state as a model *m*, assigning true/false to symbols
  3. Every clause in the original *KB* is true in *m*

     $a_1 \wedge \ldots \wedge a_k \Rightarrow b$
  4. Hence *m* is a model of *KB*
  5. If $KB \models q$, *q* is true in every model of *KB*, including *m*

Backward chaining

Idea: work backwards from the query q:

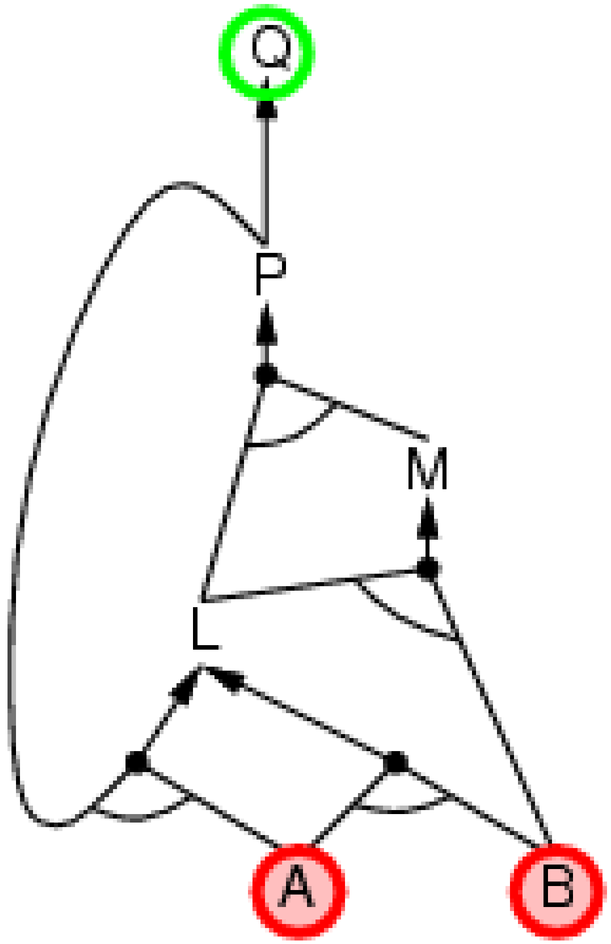
- to prove q by BC,
  - check if q is known already, or
  - prove by BC all premises of some rule concluding q

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

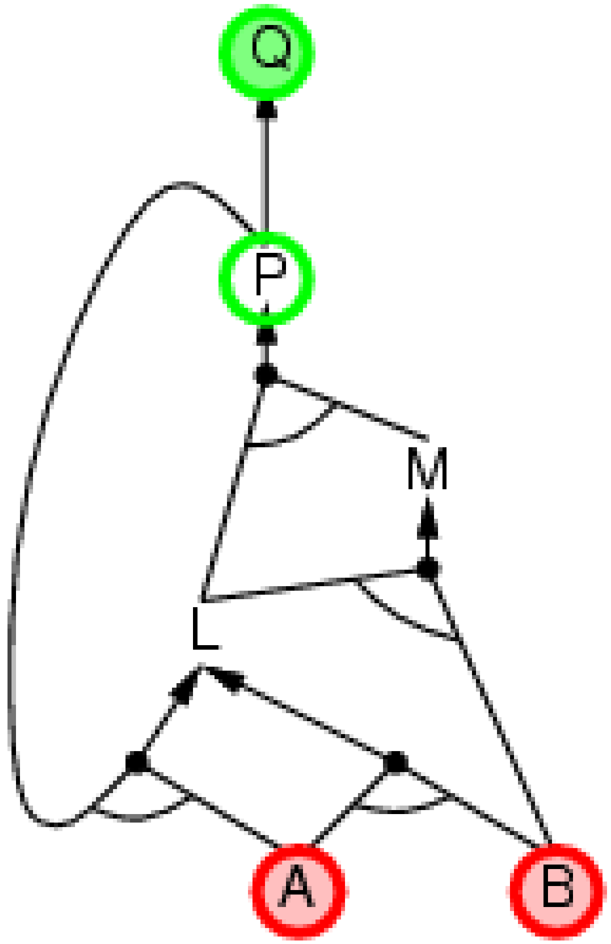
Avoid repeated work: check if new subgoal

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

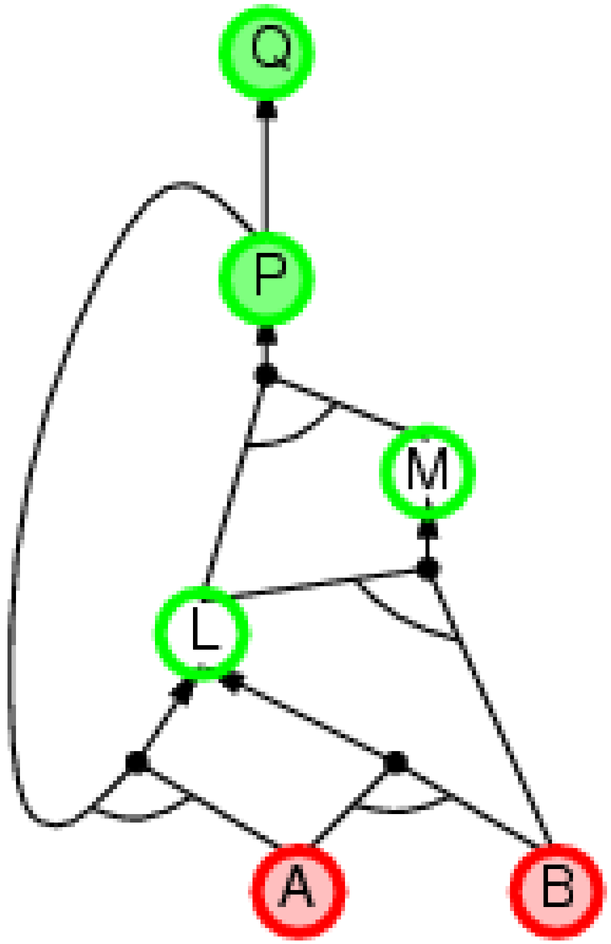
Backward chaining example



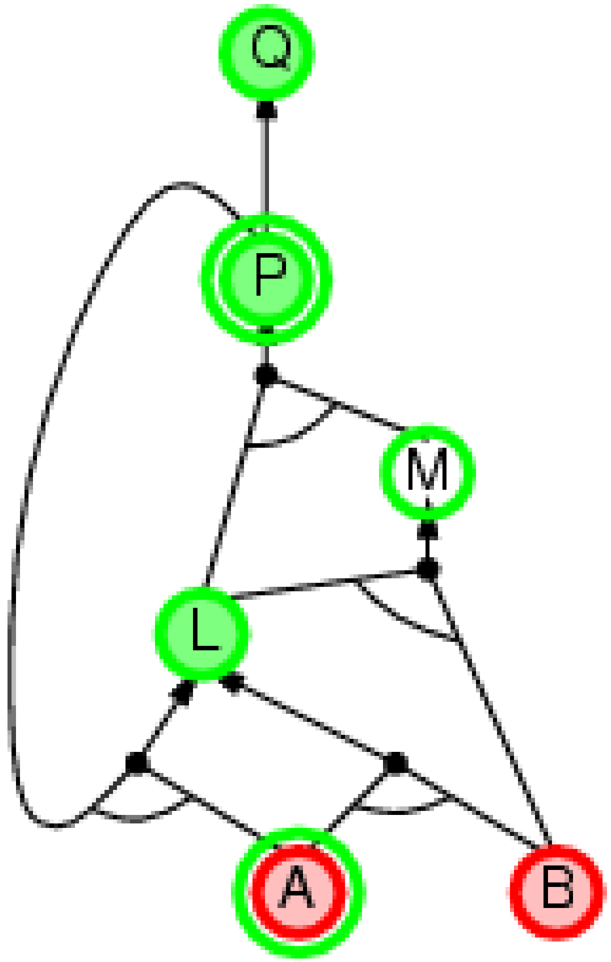
Backward chaining example



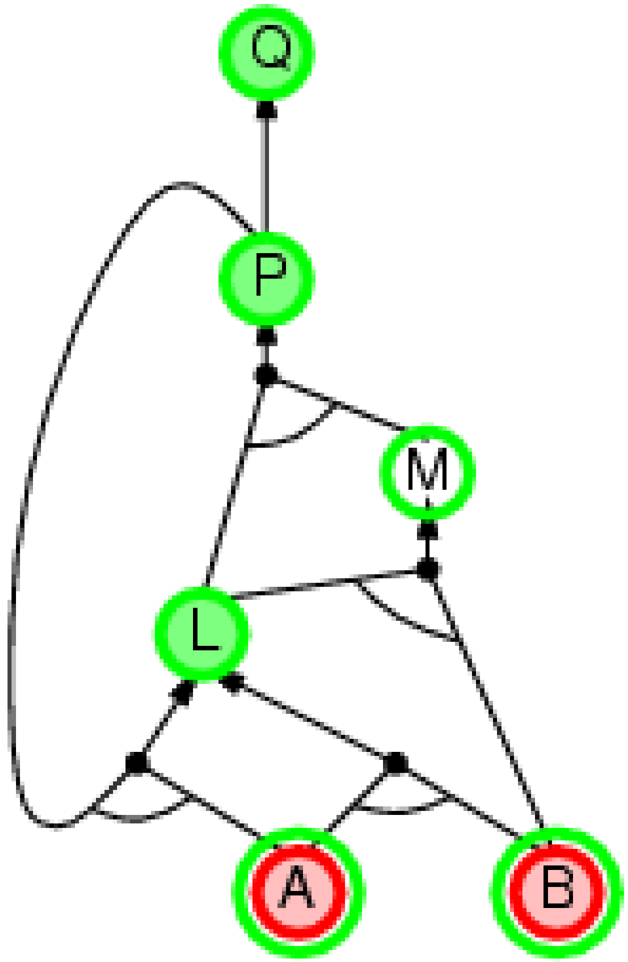
Backward chaining example



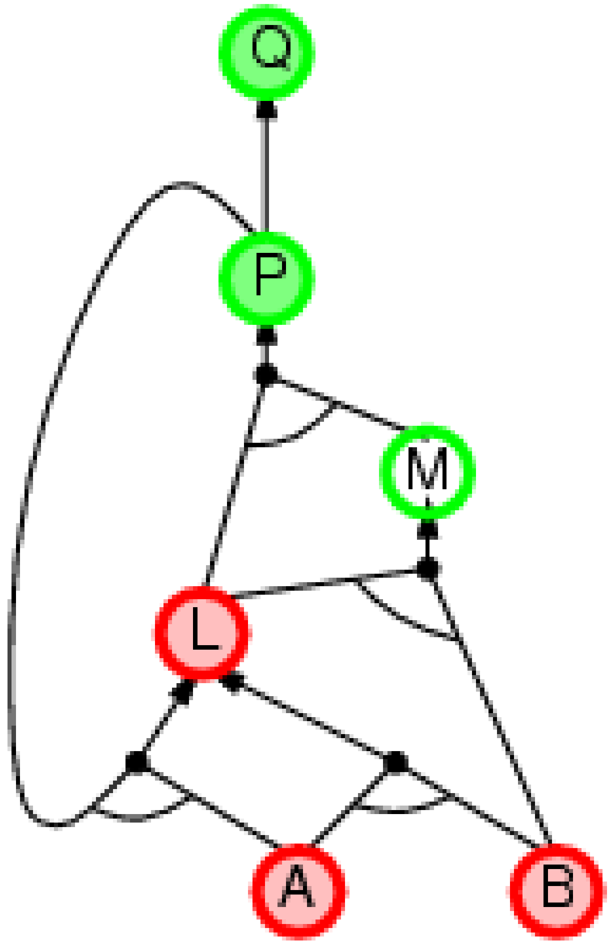
Backward chaining example



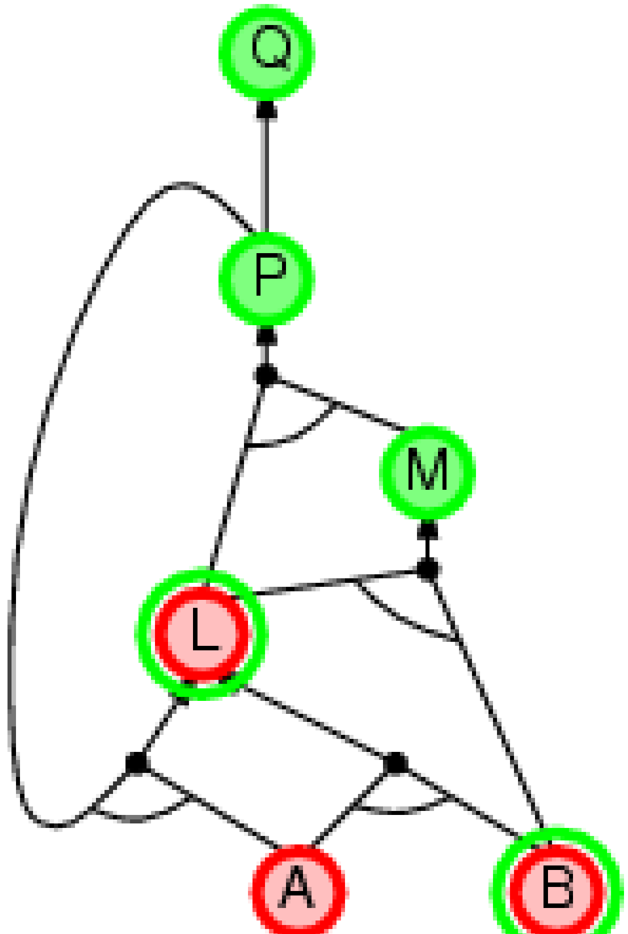
Backward chaining example



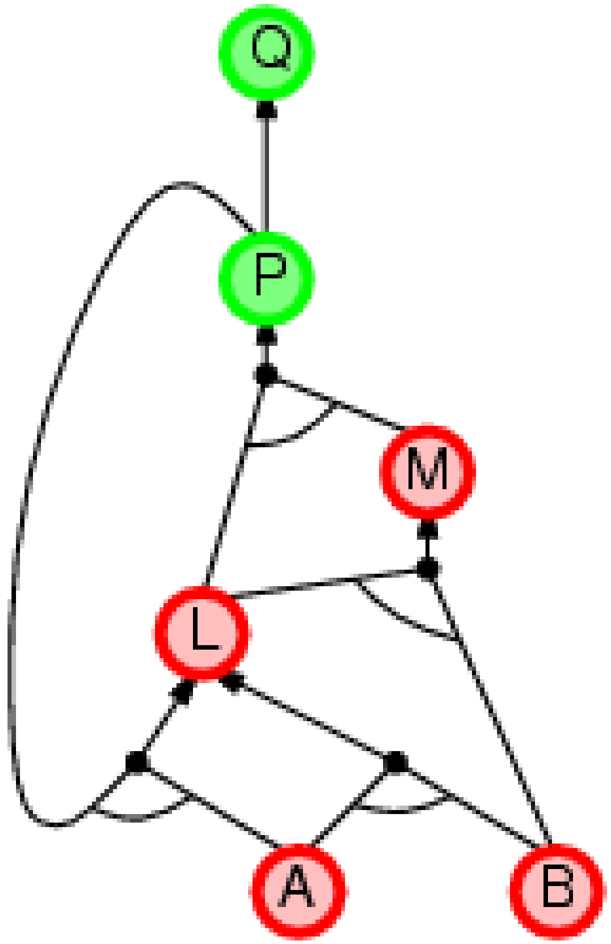
Backward chaining example



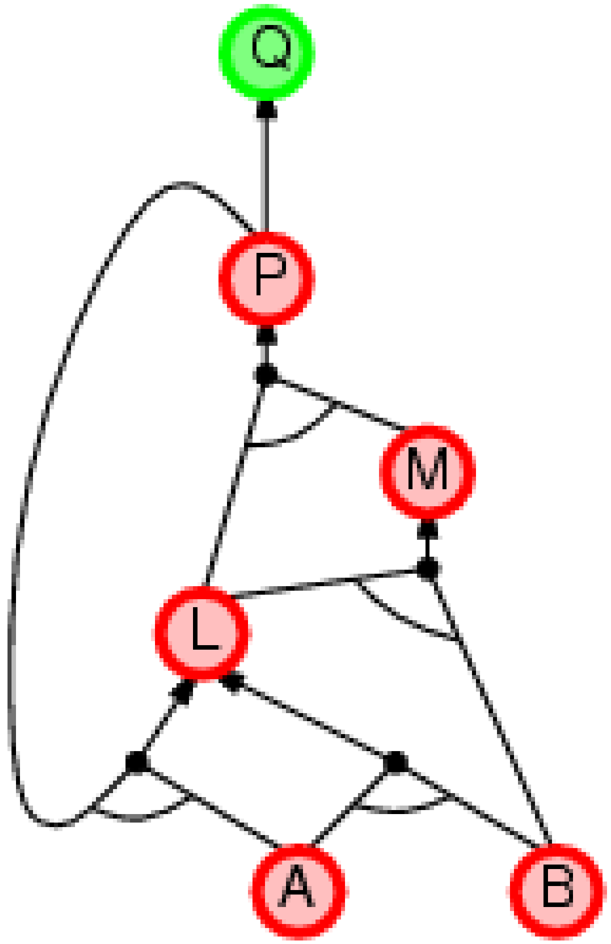
Backward chaining example



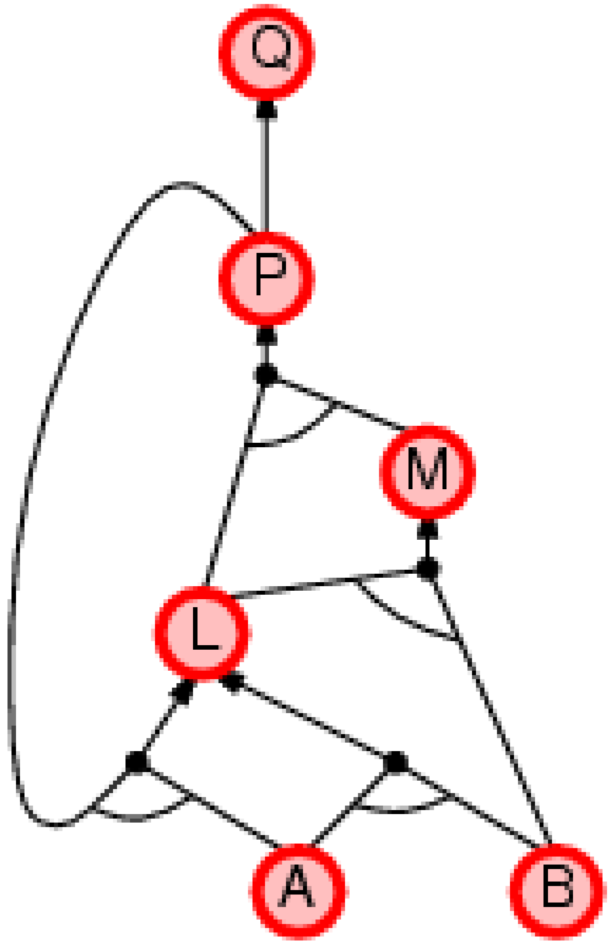
Backward chaining example



Backward chaining example



Backward chaining example



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

   Make a pure symbol literal true.

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.,

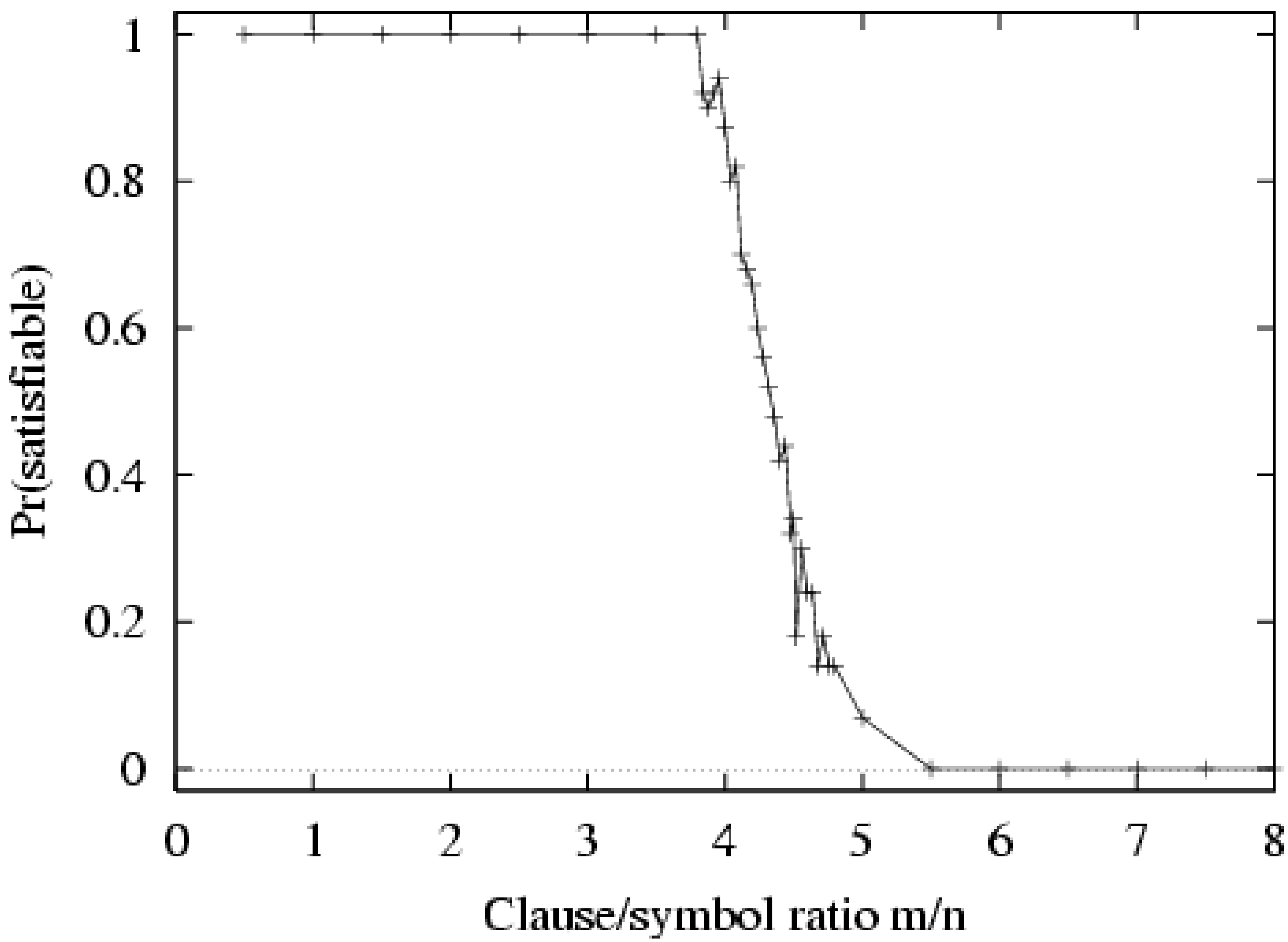
  $(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$

  m = number of clauses

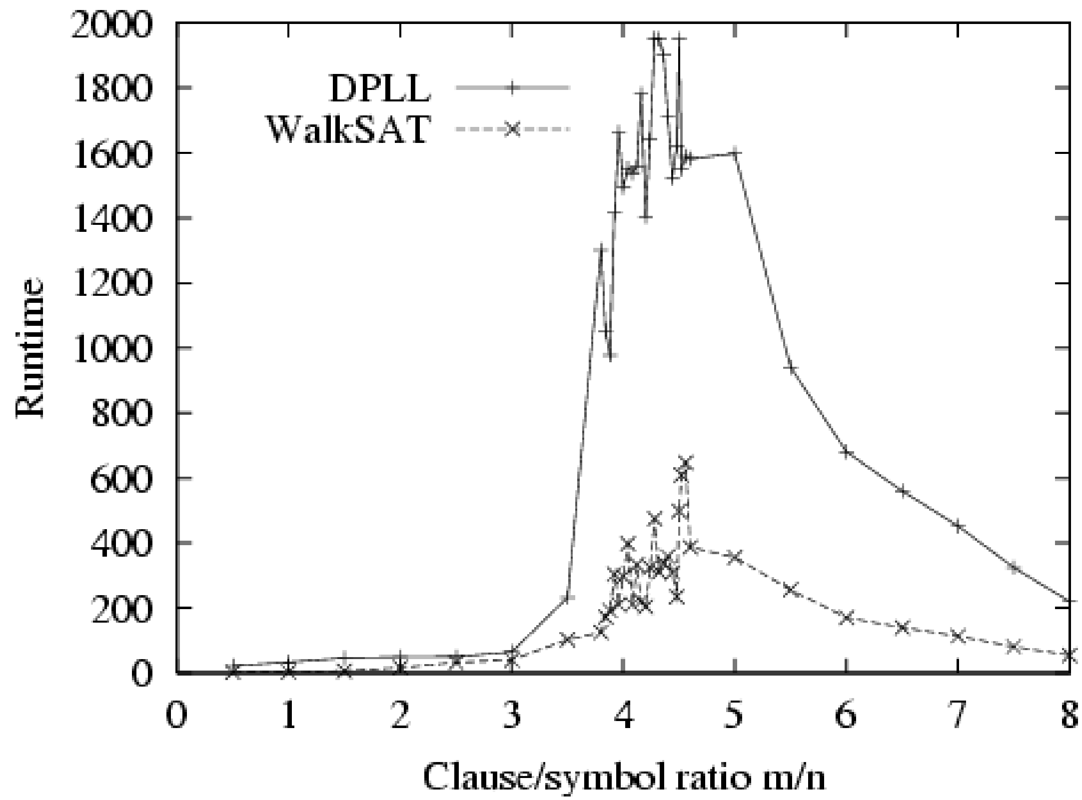
  n = number of symbols

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

Hard satisfiability problems



Hard satisfiability problems



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

Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

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

$\Rightarrow$ 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
```

Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

- For every time t and every location $[x,y]$,

$$L^t_{x,y} \wedge FacingRight^t \wedge Forward^t \Rightarrow L^t_{x+1,y}$$

- Rapid proliferation of clauses

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.
- Resolution is complete for propositional logic
  Forward, backward chaining are linear-time, complete for Horn clauses
- Propositional logic lacks expressive power