ARTIFICIAL INTELLIGENCE

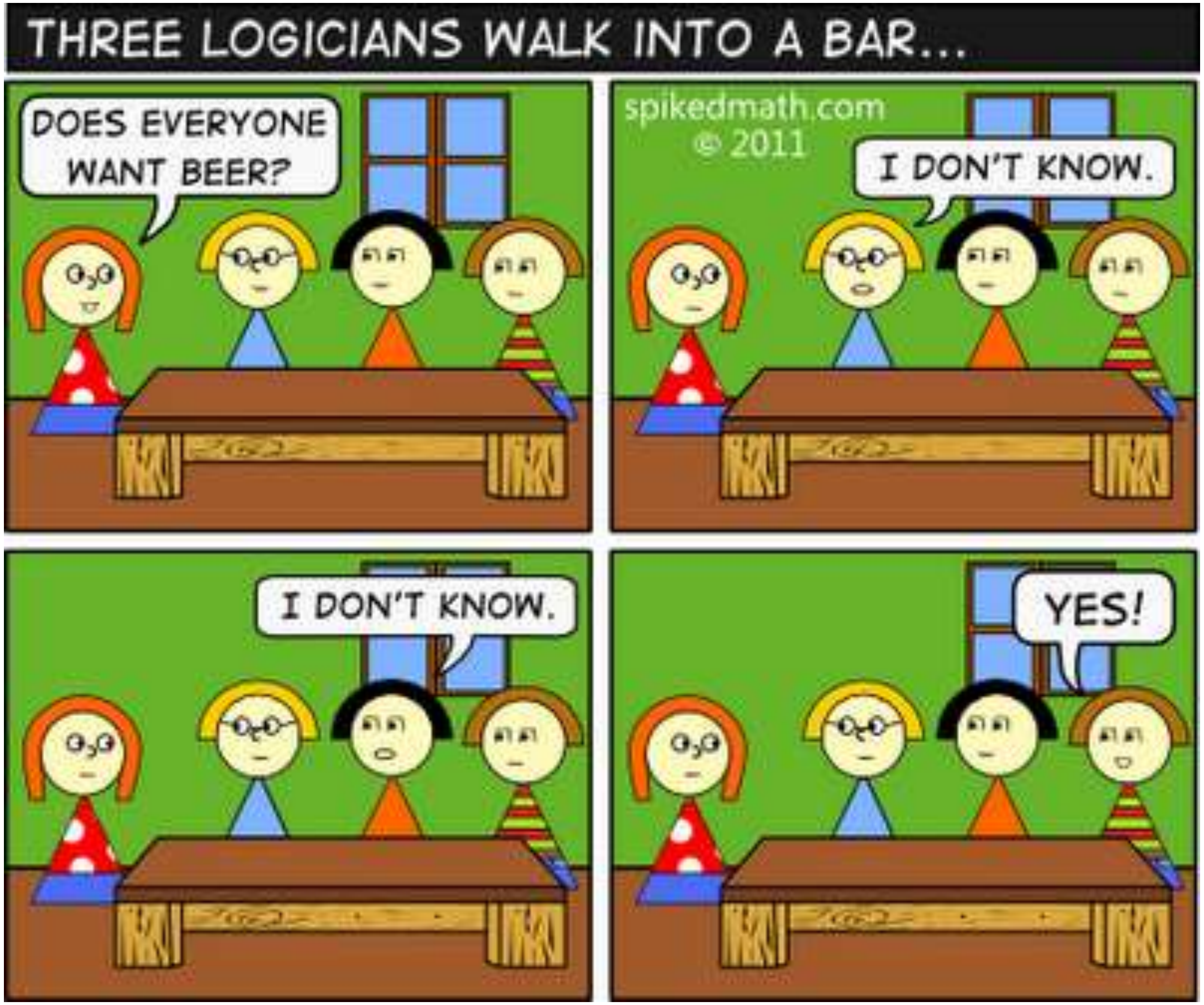
2023/2024 Semester 1

Logical Agents: Chapter 7

Problems in AI

- Problem Formulation
- Uninformed Search
- Heuristic Search
- Adversarial Search (Multi-agents)
- **Knowledge Representation**
- Rule-Based Inference and Learning
- Uncertainty

Logic



THREE LOGICIANS WALK INTO A BAR...
DOES EVERYONE WANT BEER?
spikedmath.com © 2011
I DON'T KNOW.
I DON'T KNOW.
YES!

A story

- You roommate comes home; he/she is completely wet
- You know the following things:
  - Your roommate is wet
  - If your roommate is wet, it is because of rain, sprinklers, or both
  - If your roommate is wet because of sprinklers, the sprinklers must be on
  - If your roommate is wet because of rain, your roommate must not be carrying the umbrella
  - The umbrella is not in the umbrella holder
  - If the umbrella is not in the umbrella holder, either you must be carrying the umbrella, or your roommate must be carrying the umbrella
  - You are not carrying the umbrella
- Can you conclude that the sprinklers are on?
- Can AI conclude that the sprinklers are on?

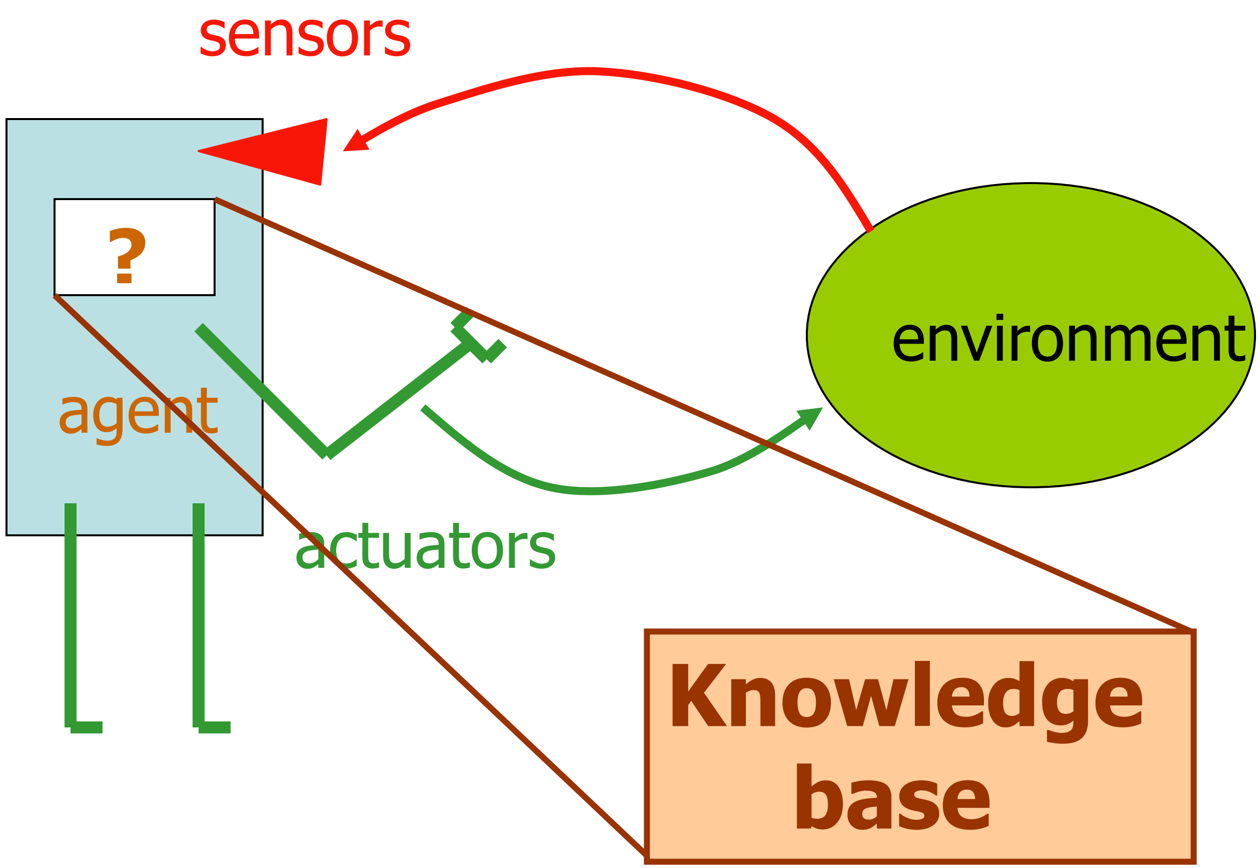
Knowledge base for the story

- RoommateWet
- RoommateWet => (RoommateWetBecauseOfRain OR RoommateWetBecauseOfSprinklers)
- RoommateWetBecauseOfSprinklers => SprinklersOn
- RoommateWetBecauseOfRain => NOT(RoommateCarryingUmbrella)
- UmbrellaGone
- UmbrellaGone => (YouCarryingUmbrella OR RoommateCarryingUmbrella)
- NOT(YouCarryingUmbrella)

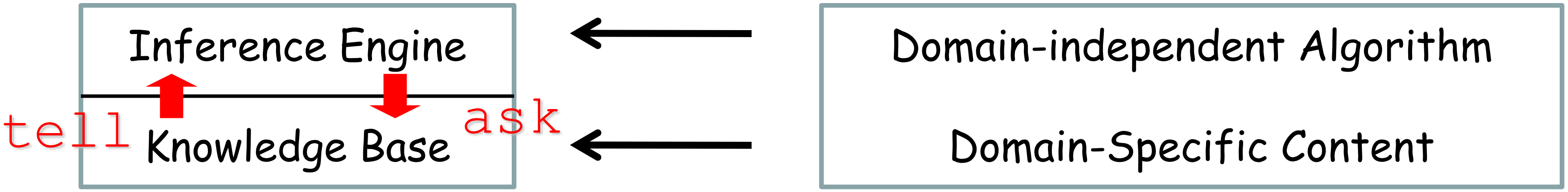
Outline

- **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**

Knowledge-based agent



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

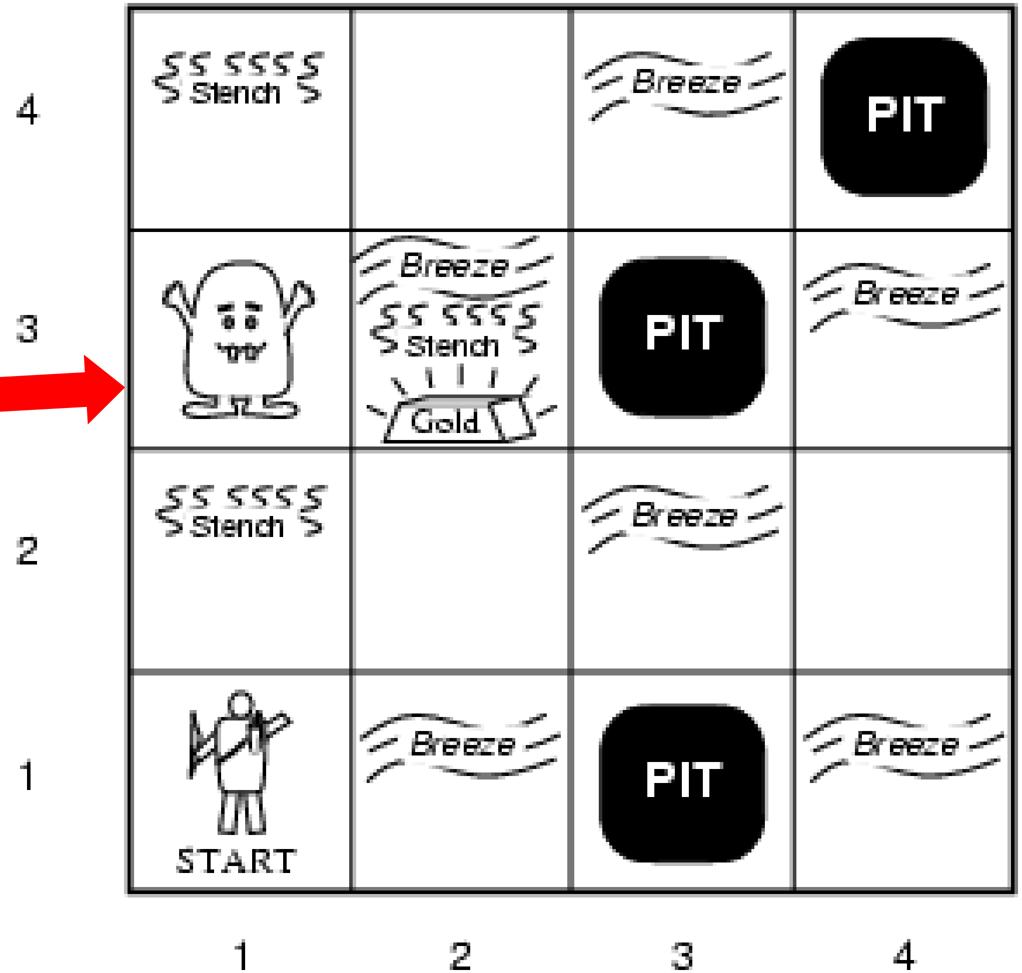
```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    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

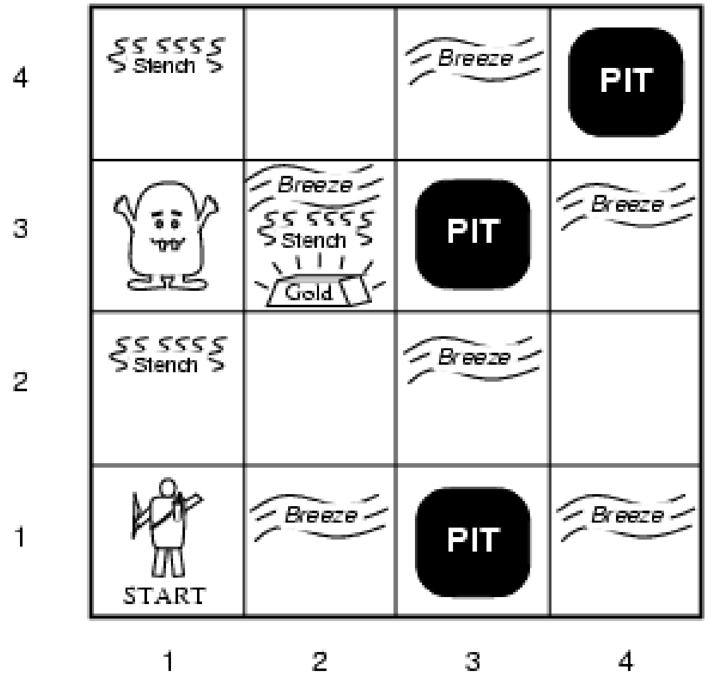
Wumpus World

Wumpus



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 scream
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square
  - You bump if you walk into a wall
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot
- Sensors: Stench, Breeze, Glitter, Bump, Scream
-

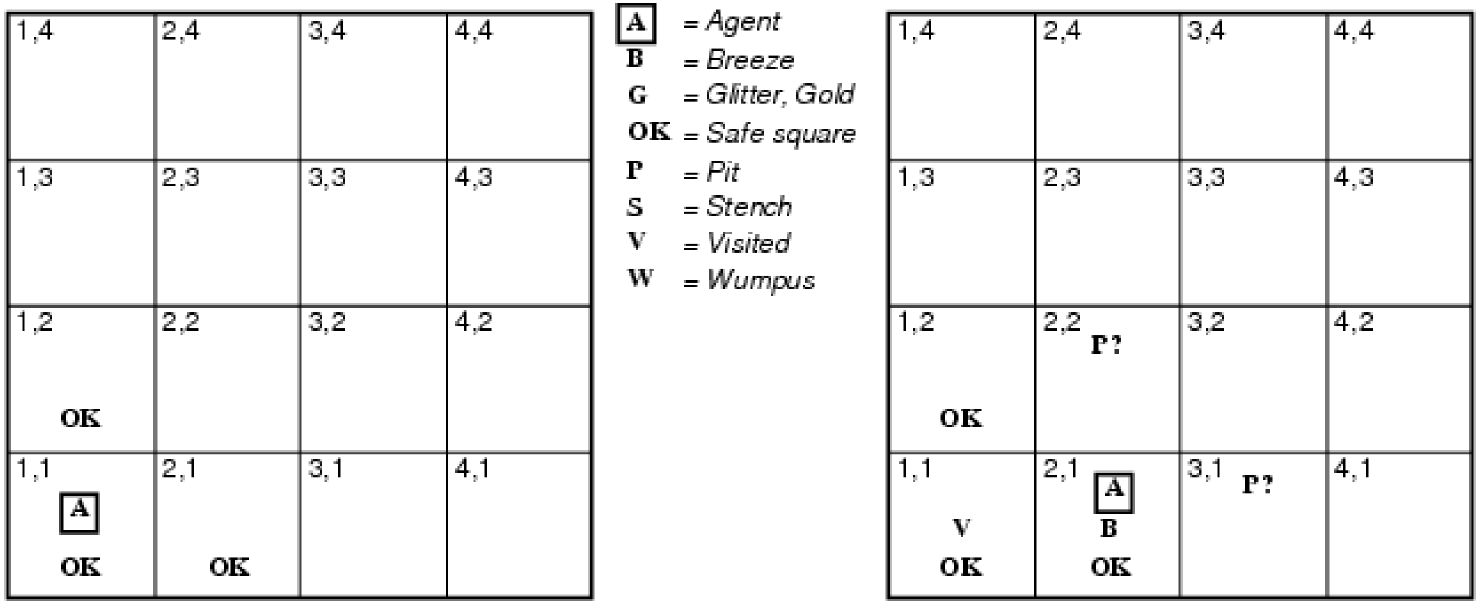


Wumpus world characterization

- 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



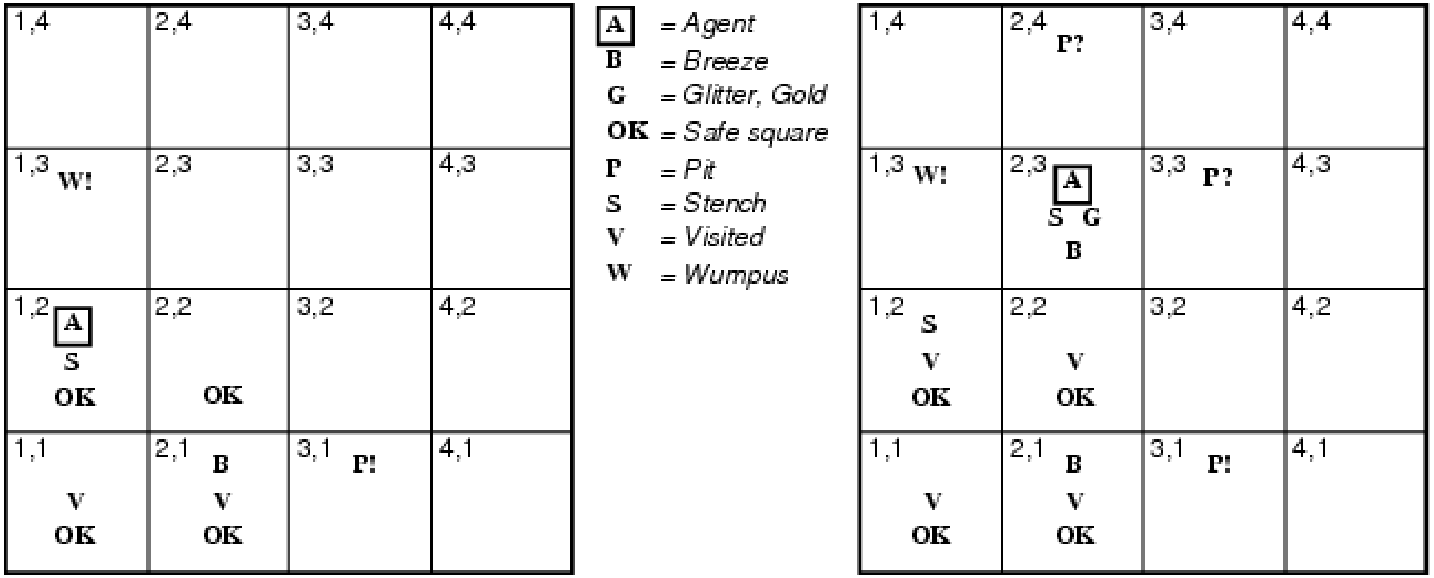
(a) (b)

1. The KB initially contains the rules of the environment.

2. [1,1] The first percept is [*none*, *none*,*none*,*none*,*none*],
   Move to safe cell e.g. 2,1

3. [2,1] Breeze 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: wumpus is in [1,3] or [2,2]

YET … not in [1,1]

Thus … 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 [3,1]

Move to next safe cell [2,2]

Exploring the Wumpus World

(a) (b)

[2,2] Detect nothing

Move to unvisited safe cell e.g. [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

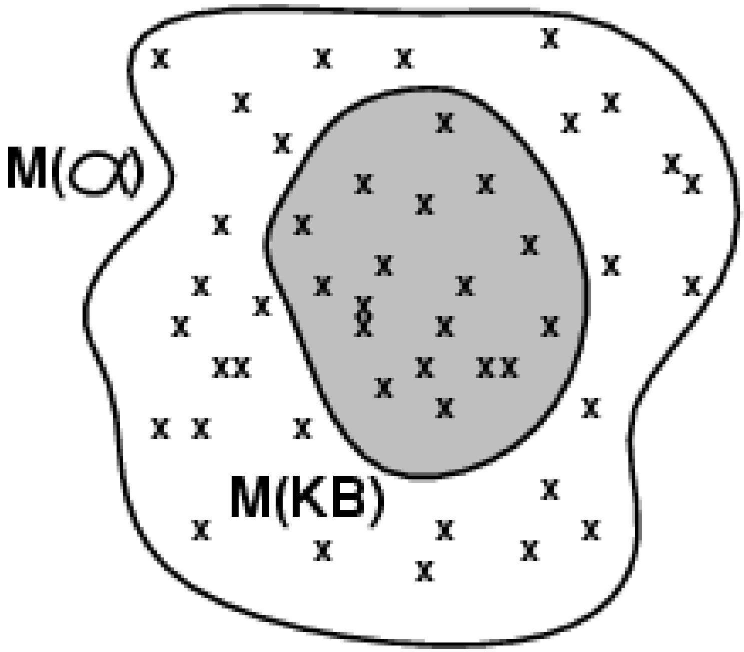
- Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

- 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

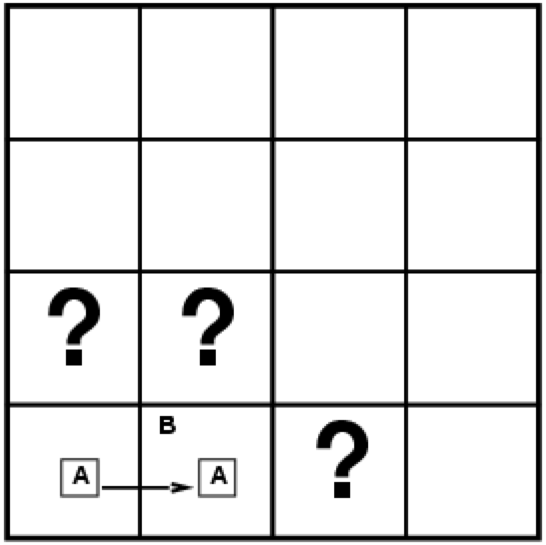
- Then KB $\models$ α iff *M(KB)* $\subseteq$ *M*(α)

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



Entailment in the wumpus world

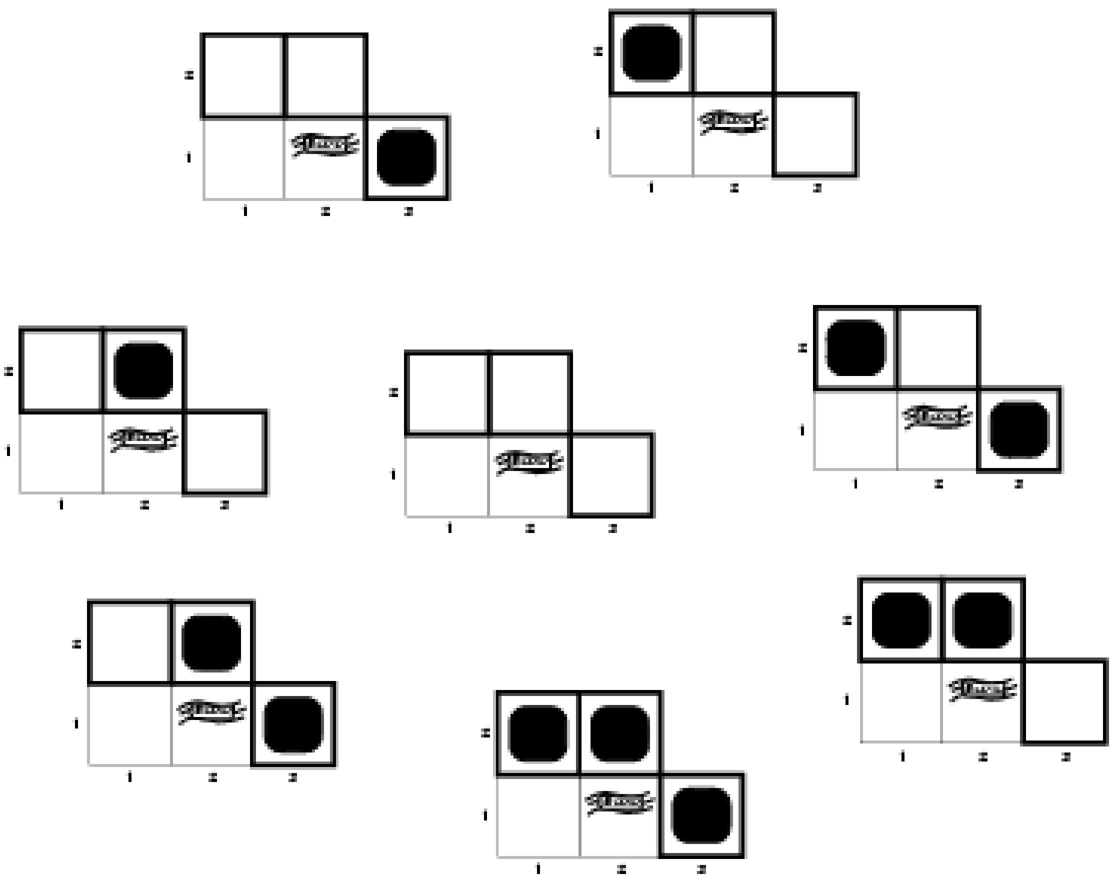
- Situation after detecting nothing in [1,1], moving right, breeze in [2,1]



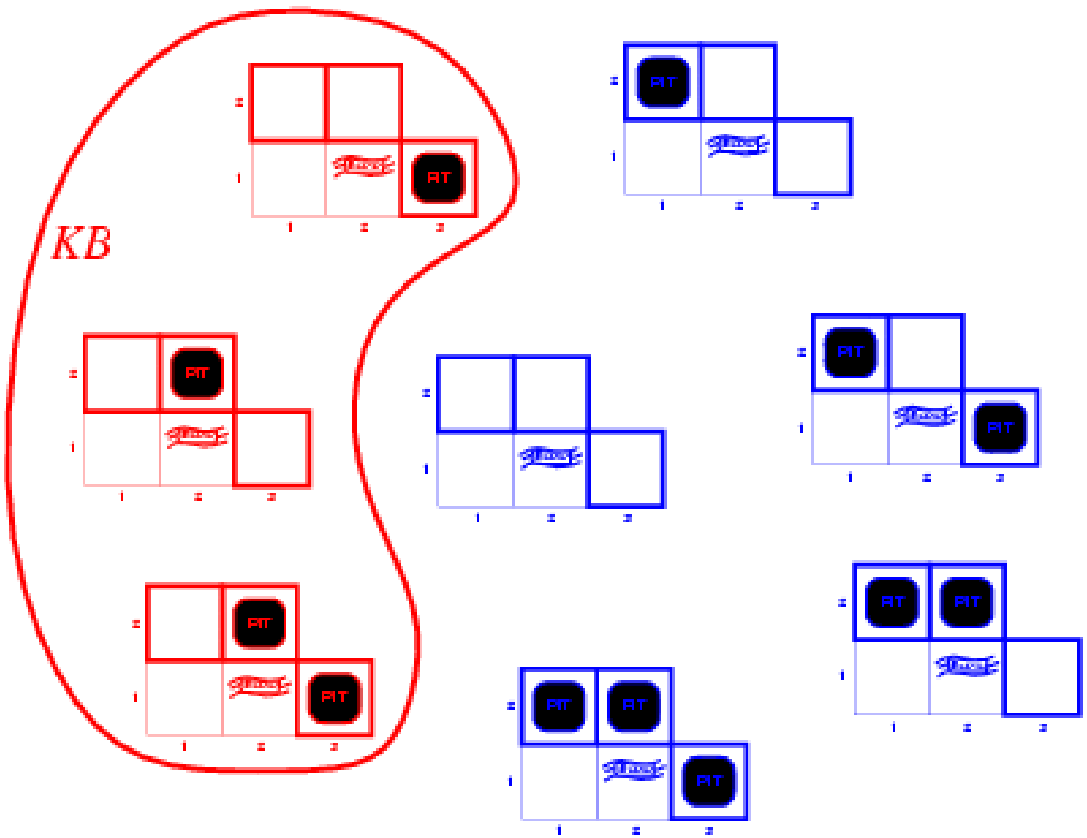
- Consider possible models for *KB* assuming only pits

- Boolean choices ⇒ 8 possible models

Wumpus models

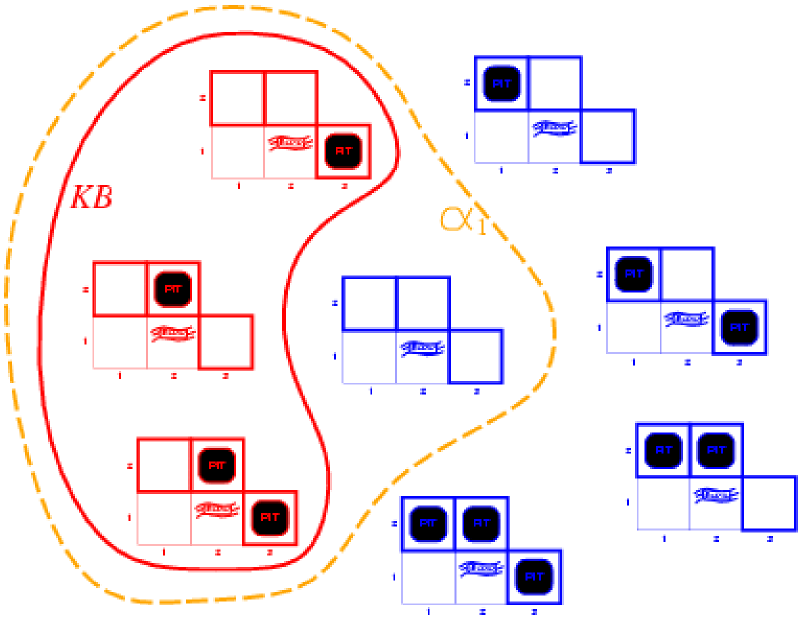


Wumpus models



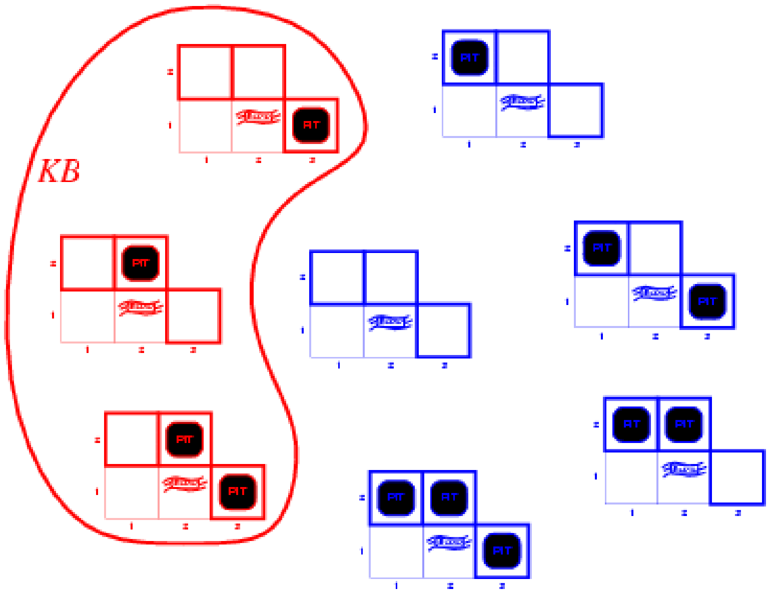
- *KB* = wumpus-world rules + observations

Wumpus models



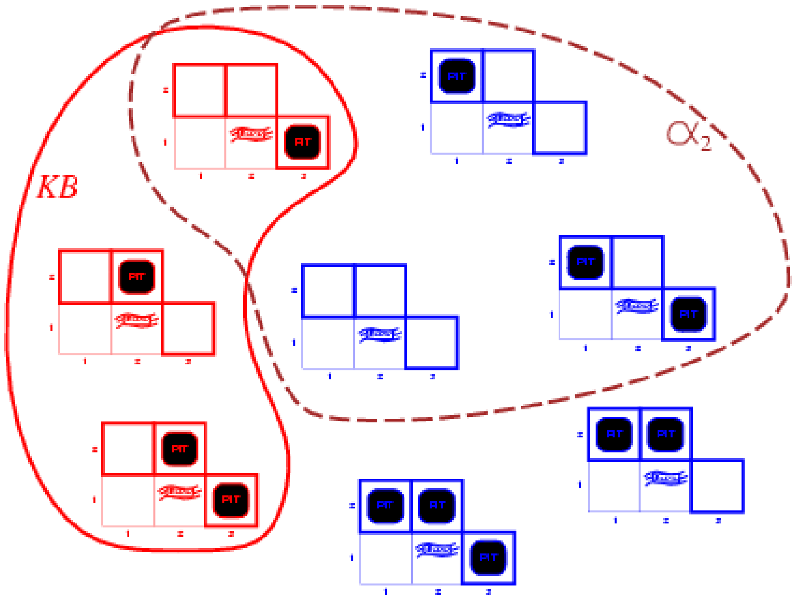
- *KB* = wumpus-world rules + observations
- α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by model checking

Wumpus models



- *KB* = wumpus-world rules + observations

Wumpus models



- *KB* = wumpus-world rules + observations
- α_2 = "[2,2] is safe", $KB \models \alpha_2$?   no
-

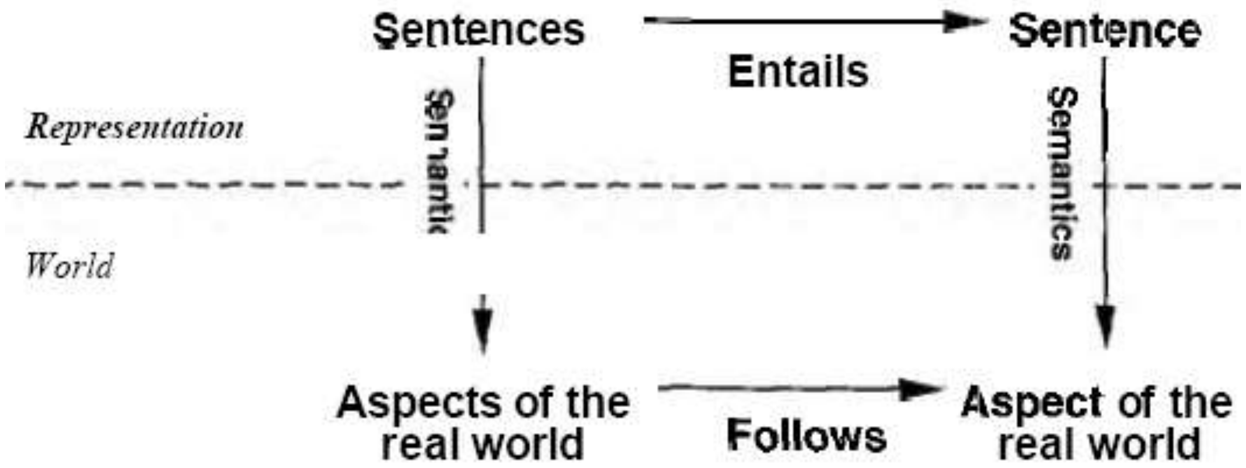
Property of inference algorithm

- An inference algorithm that derives only entailed sentences is called **sound or truth-preserving.**

- An inference algorithm is **complete** if it can derive any sentence that is entailed.

Property of inference algorithm

- **if *KB is true in the real world, then any sentence Alpha derived from KB by a sound inference procedure is also true in the real world.***
- The final issue that must be addressed by an account of logical agents is that of **grounding**-the connection, if any, between logical reasoning processes and the real environment in which the agent exists.
- Sensors and **learning**



Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas
- The proposition symbols P_1, P_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: Syntax

A BNF grammar of sentences in propositional logic:

$$
\begin{aligned}
\mathit{Sentence} &\rightarrow \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} &\rightarrow \textbf{True} \mid \textbf{False} \mid \text{Symbol} \\
\mathit{Symbol} &\rightarrow P \mid \mathbf{Q} \mid \mathbf{R} \mid \ldots \\
\mathit{ComplexSentence} &\rightarrow \neg\, \mathit{Sentence} \\
&\mid\ (\, \mathit{Sentence} \wedge \mathit{Sentence}\,) \\
&\mid\ (\, \mathit{Sentence} \vee \mathit{Sentence}\,) \\
&\mid\ (\, \mathit{Sentence} \Rightarrow \mathit{Sentence}\,) \\
&\mid\ (\, \mathit{Sentence} \Leftrightarrow \mathit{Sentence}\,)
\end{aligned}
$$

from highest to lowest: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$

Propositional logic: Semantics

five connectives. Atomic sentences are easy:

- *True* is true in every model and False is false in every model.
- The truth value of every other proposition symbol must be specified directly in the model. For example, in the model m_1 given earlier, $P_{1,2}$ is false.

For complex sentences, we have rules such as

- For any sentence s and any model m, the sentence $\neg s$ is true in m if and only if s is false in m.

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.,

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

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“

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

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

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

- Depth-first enumeration of all models is sound and complete
-

```
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:
  $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

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

Validity and satisfiability

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

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

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., $A \vee B$, C

A sentence is unsatisfiable if it is true in no models

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

Satisfiability is connected to inference via the following:

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

$\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg \beta)$ 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

Inference Rules

- **Modus Ponens:**

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

- **And-Elimination:**

$$\frac{\alpha \wedge \beta}{\alpha}$$

- **Other rule:**

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{and} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

The preceding derivation a sequence of applications of inference rules is called a proof.

Finding proofs is exactly like finding solutions to search problems.

Searching for proofs is an alternative to enumerating models.

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

Then we apply And-Elimination to R_6 to obtain

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

Logical equivalence for contrapositives gives

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

Now we can apply Modus Ponens with R_8 and the percept R_4 (i.e., $\neg B_{1,1}$), to obtain

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

Finally, we apply De Morgan's rule, giving the conclusion

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

That is, neither [1,2] nor [2,1] contains a pit.

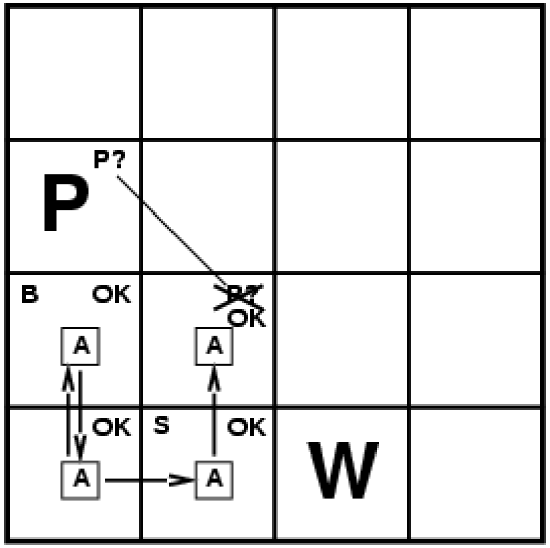
Resolution

- Resolution inference rule (for CNF):

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

where l_i and m_j are complementary literals.

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

Conversion to CNF

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literals clauses

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

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

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

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

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

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

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

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

4. Apply distributivity law ($\land$ over $\lor$) and flatten:

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

Resolution algorithm

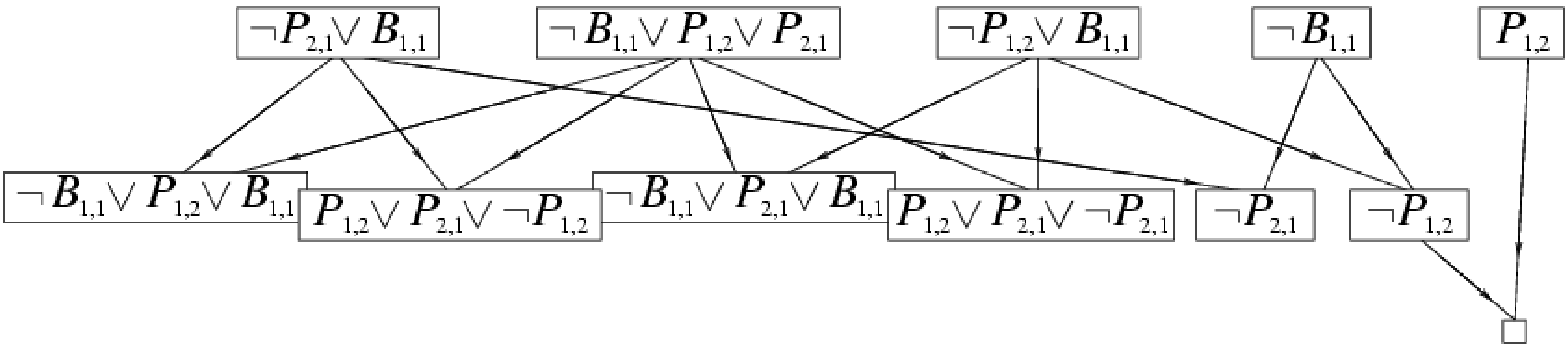
- Proof by contradiction, i.e., show *KB*∧¬α unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*

    clauses ← the set of clauses in the CNF representation of $KB \land \neg\alpha$
    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 example

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



If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Forward and backward chaining

- Horn Form (restricted)
  - KB = conjunction of Horn clauses
- **Horn clause** : a disjunction of literals of which at most one is positive
  - E.g., $\neg L_{1,1} \vee \neg \text{Breeze} \vee B_{1,1}$, $L_{1,1} \wedge \text{Breeze} \Rightarrow B_{1,1}$
  - proposition symbol, (conjunction of symbols) $\Rightarrow$ symbol
  - E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$,
  - Fact: a sentence consisting of a single positive literal, e.g. $L_{1,1}$
- Modus Ponens (for Horn Form): complete for Horn KBs

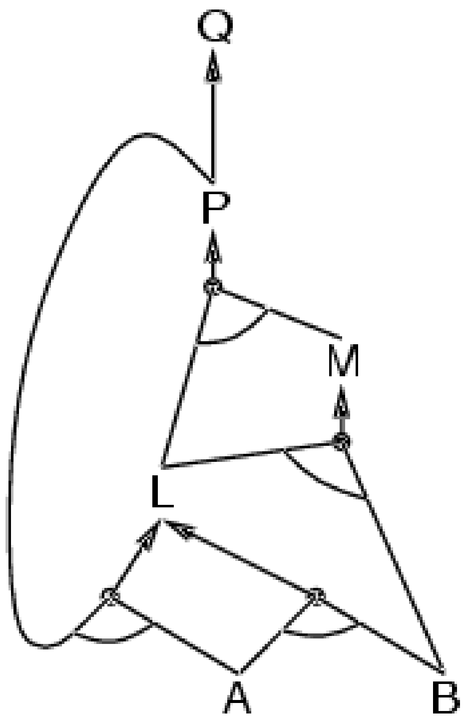
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- 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

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



Forward chaining algorithm

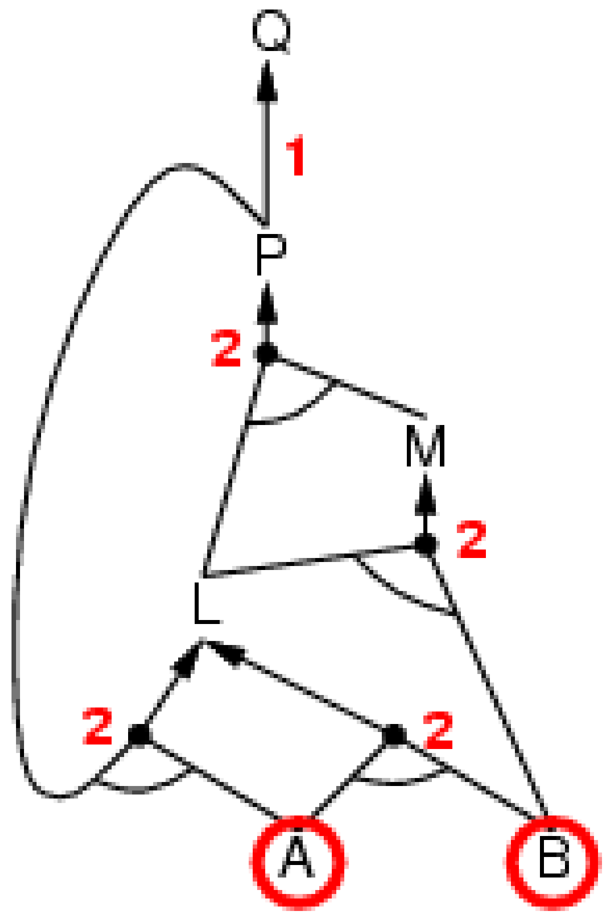
```
function PL-FC-ENTAILS?(KB, q) returns true or false
   local variables: count, a table, indexed by clause, initially the number of premises
                    inferred, a table, indexed by symbol, each entry initially false
                    agenda, a list of symbols, initially the symbols known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

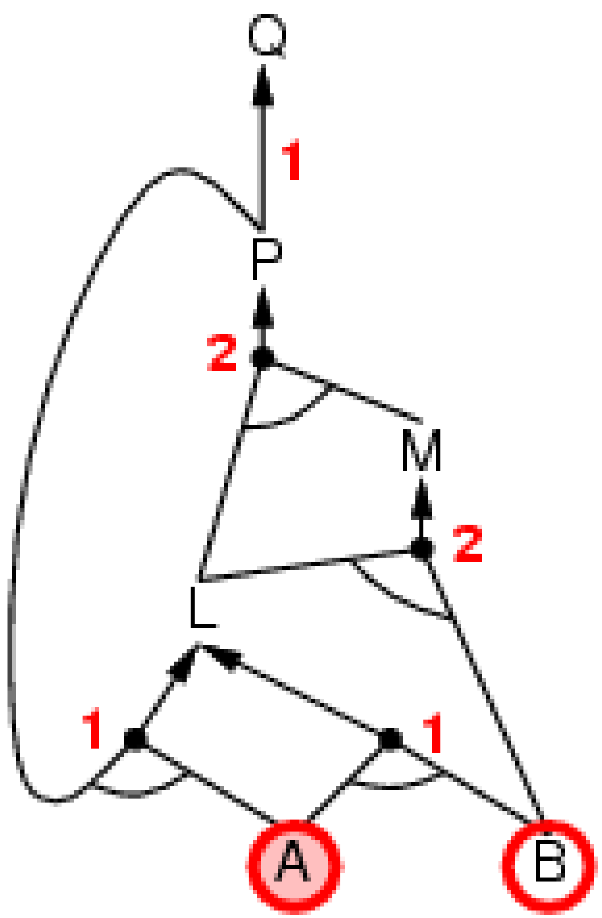
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



A
B

Forward chaining example

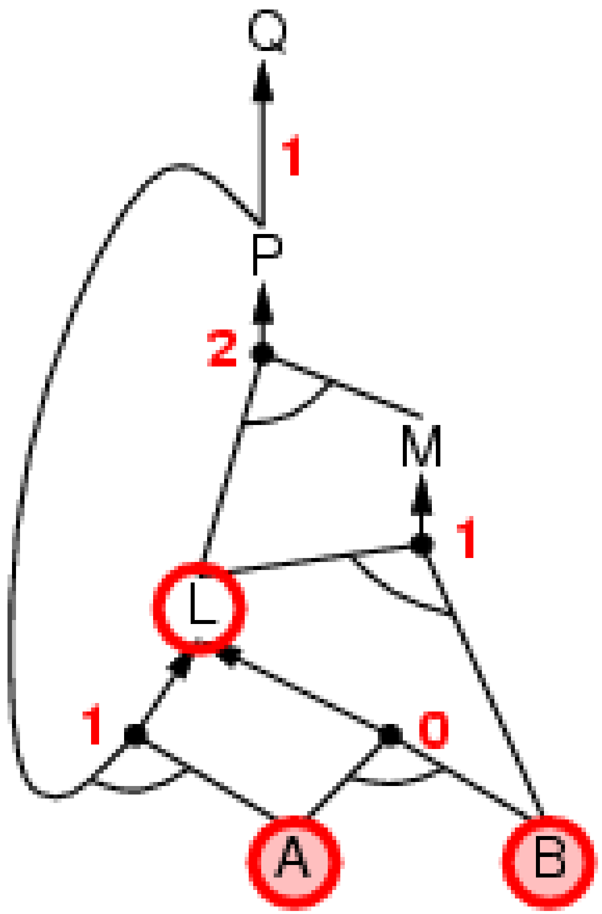
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



A ∧ ?P => ?L

Forward chaining example

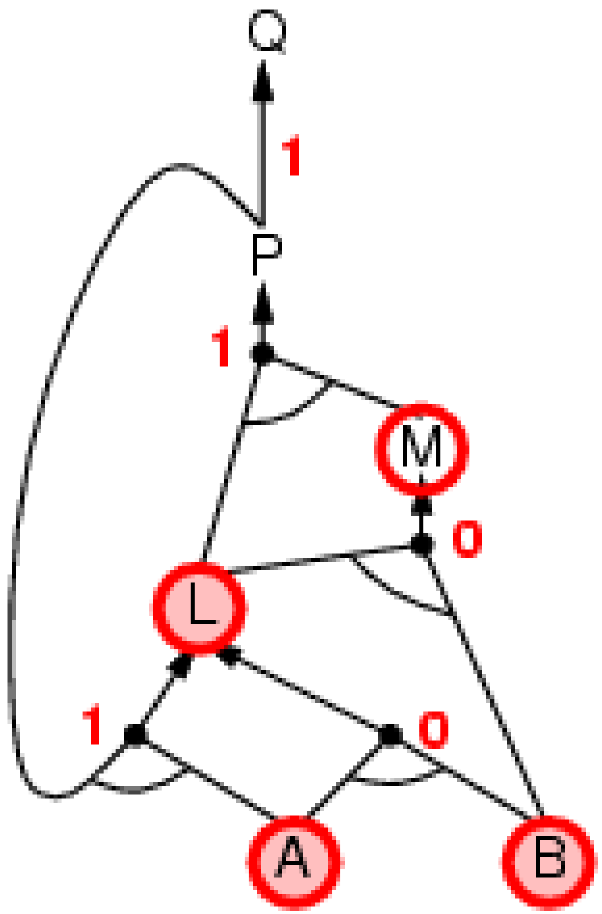
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



A ∧ *B* => *L*

Forward chaining example

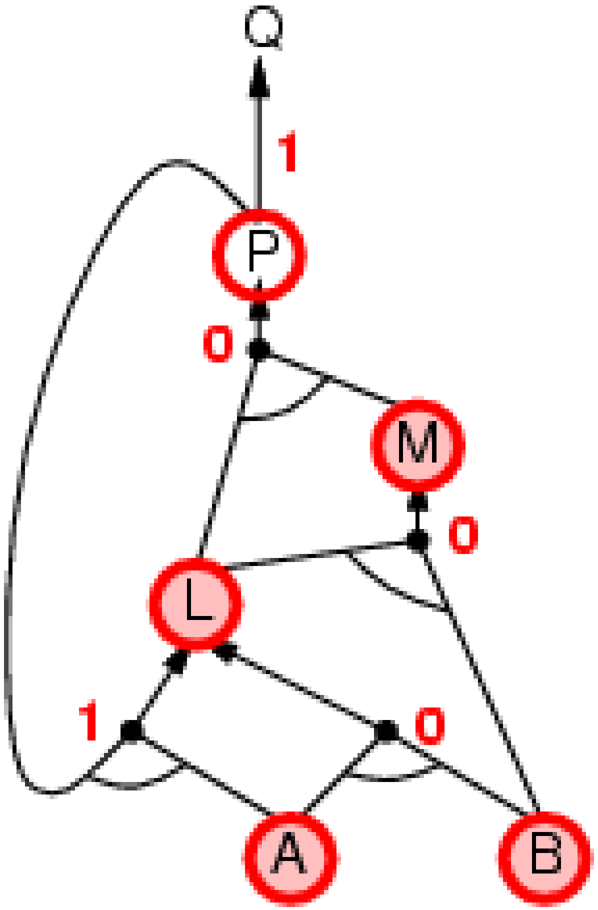
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



L ∧ *B* => *M*

Forward chaining example

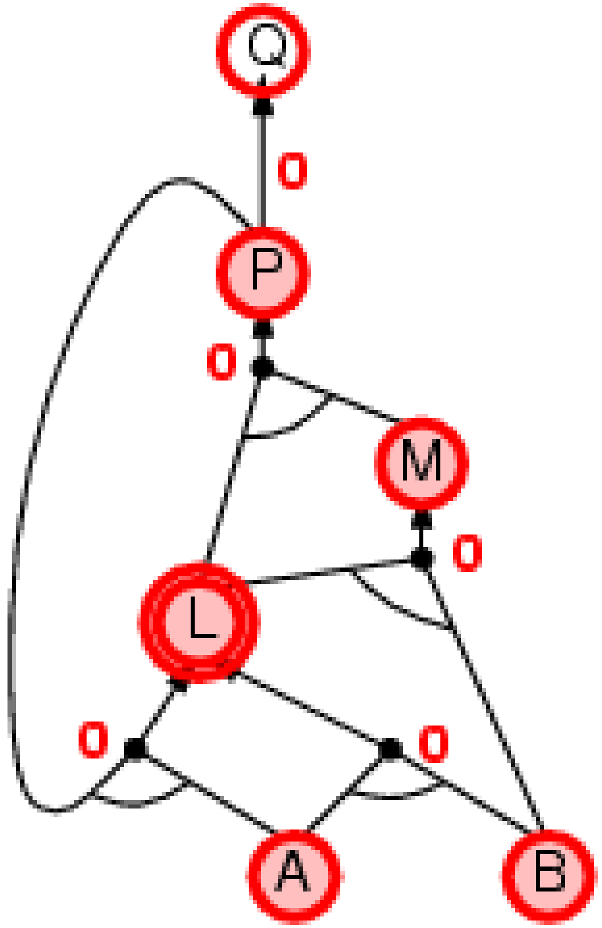
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



$M \wedge L \Rightarrow P$

Forward chaining example

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



$P \Rightarrow Q$

Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

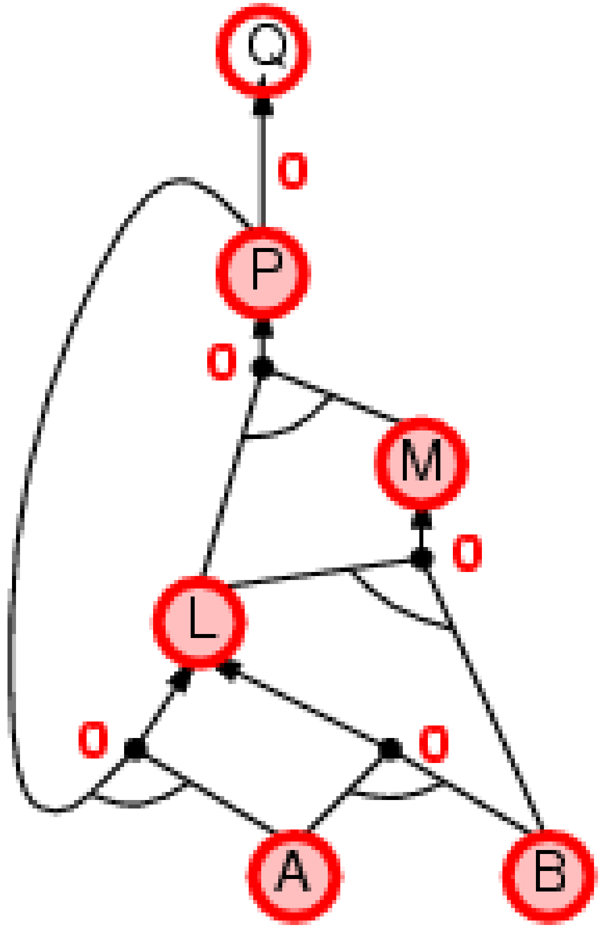
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

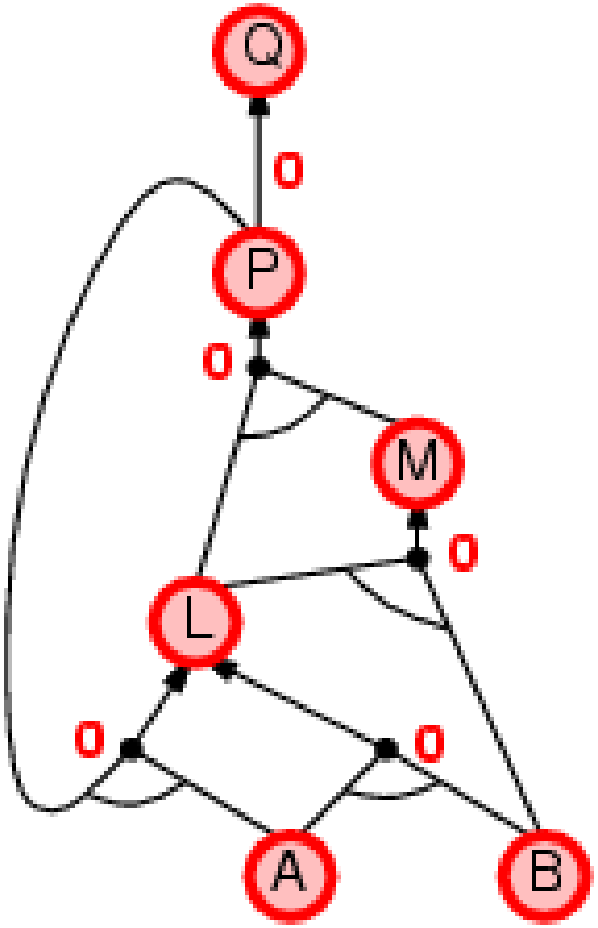
B



$A \wedge P \Rightarrow L$

Forward chaining example

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



P => Q

Backward chaining

Idea: work backwards from the query *q*:

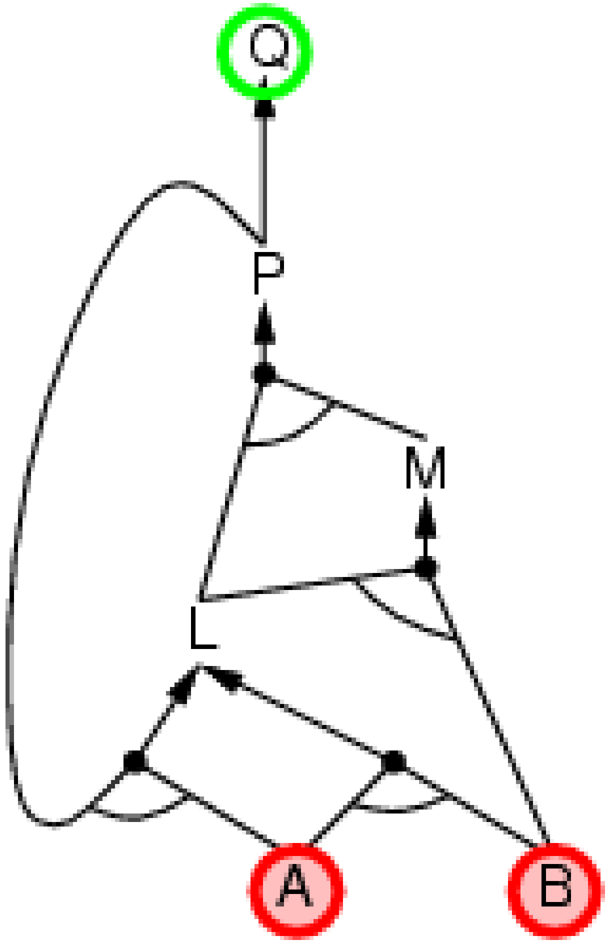
to prove *q* by BC,
check if *q* is known already, or
prove by BC all premises of some rule concluding *q*

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

Avoid repeated work: check if new sub-goal

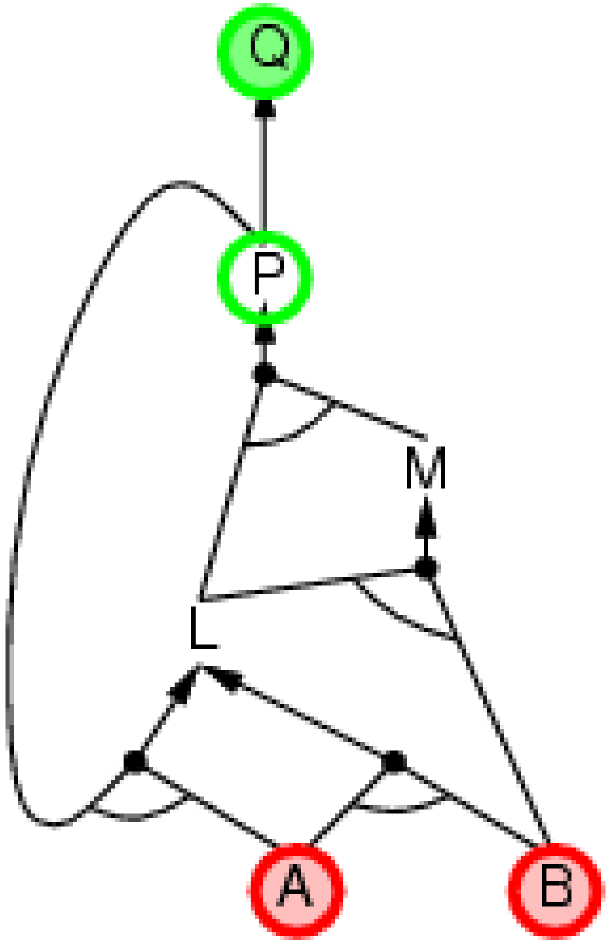
1. has already been proved true, or
2. has already failed

Backward chaining example



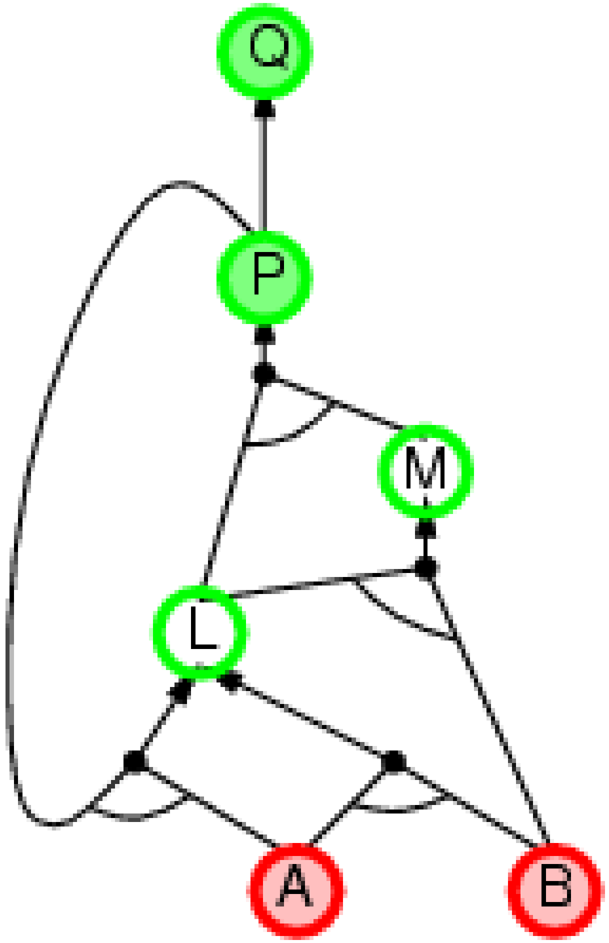
A
B
Q?

Backward chaining example



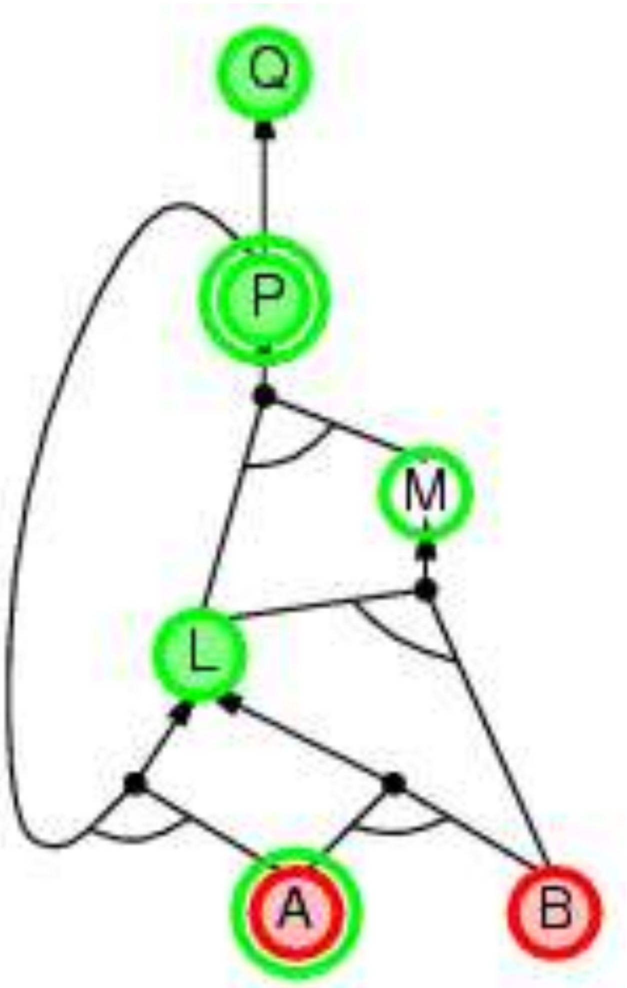
A
B
Q?<=P?

Backward chaining example



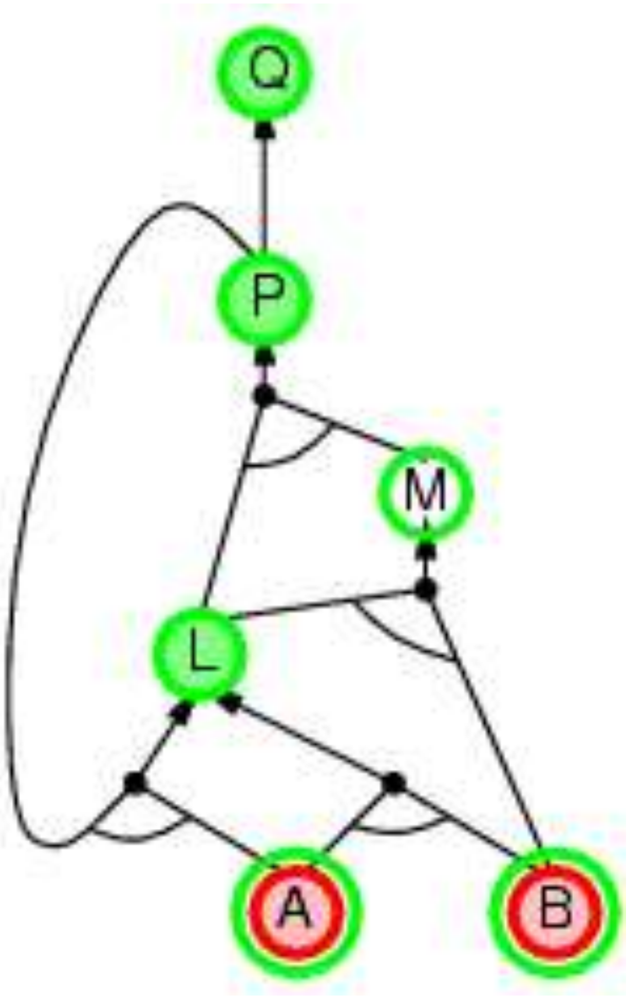
A
B
Q?<=P?
L? ∧ M? => P?

Backward chaining example



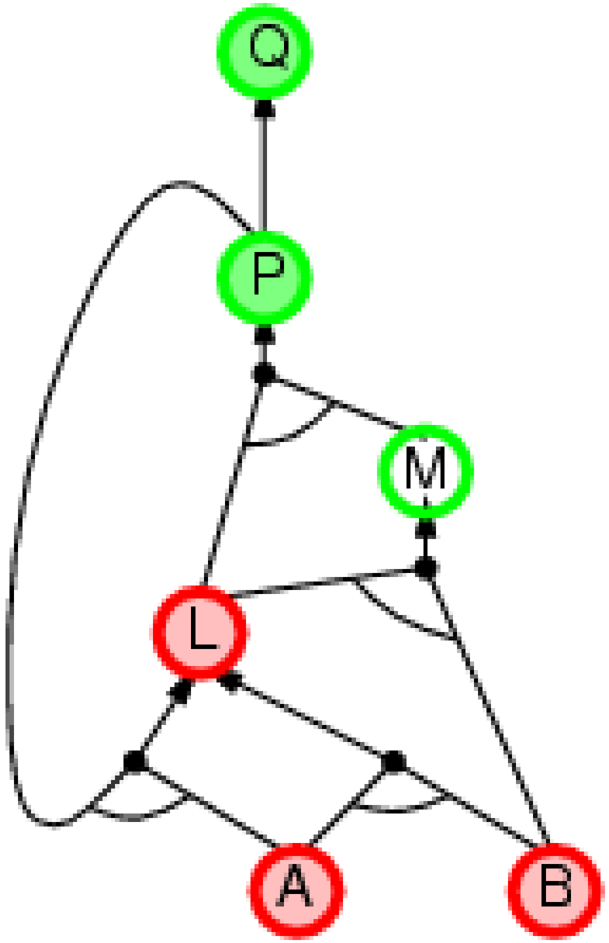
A
B
Q?<=P?
L? ∧ M? => P?
P? ∧ A => L?

Backward chaining example



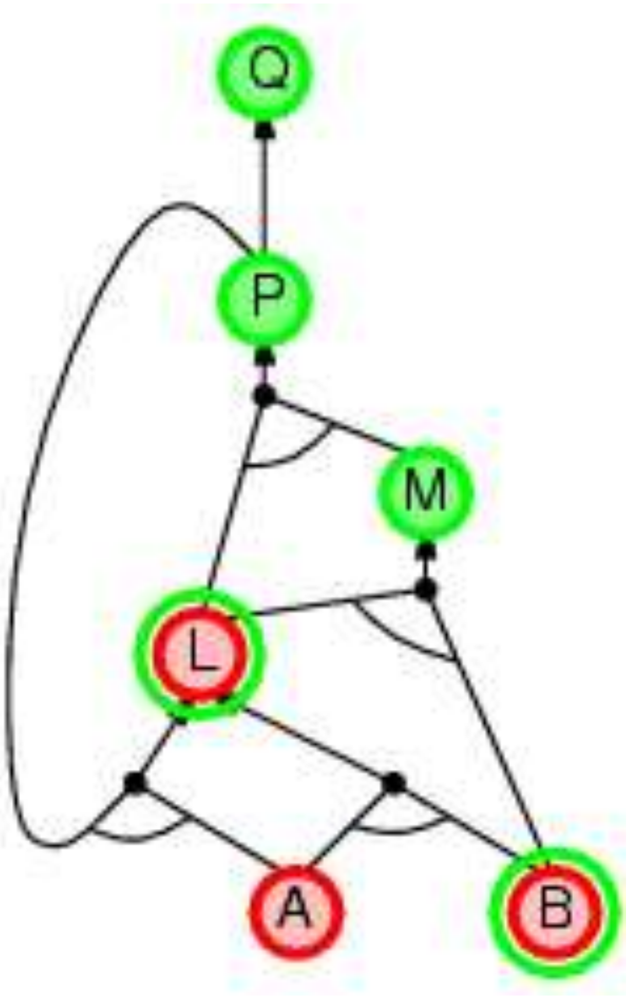
A
B
Q?<=P?
L? ∧ M? => P?
P? ∧ A => L?
A ∧ B => L

Backward chaining example



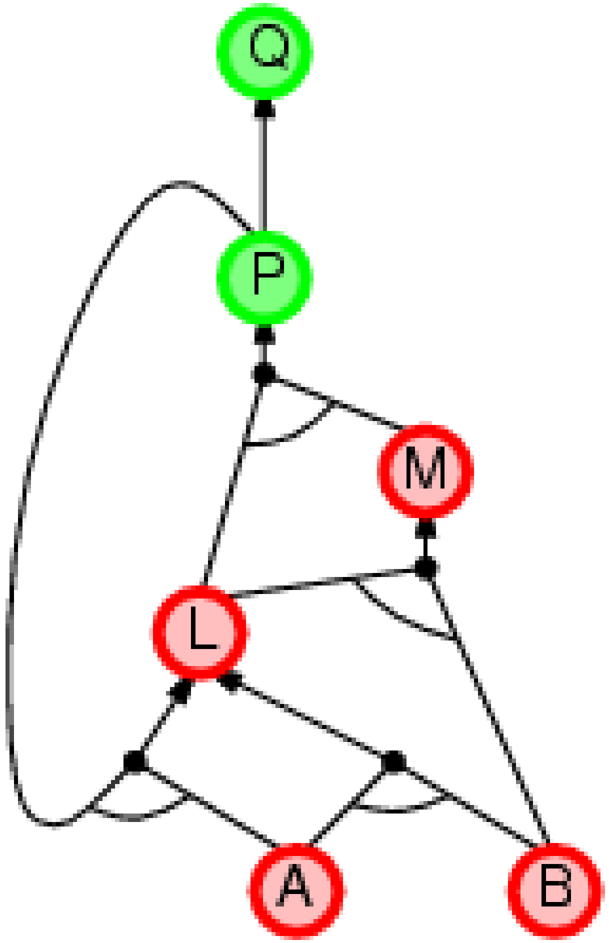
A
B
Q?<=P?
L? ∧ *M? => P?*
P? ∧ *A => L?*
A ∧ *B => L*

Backward chaining example



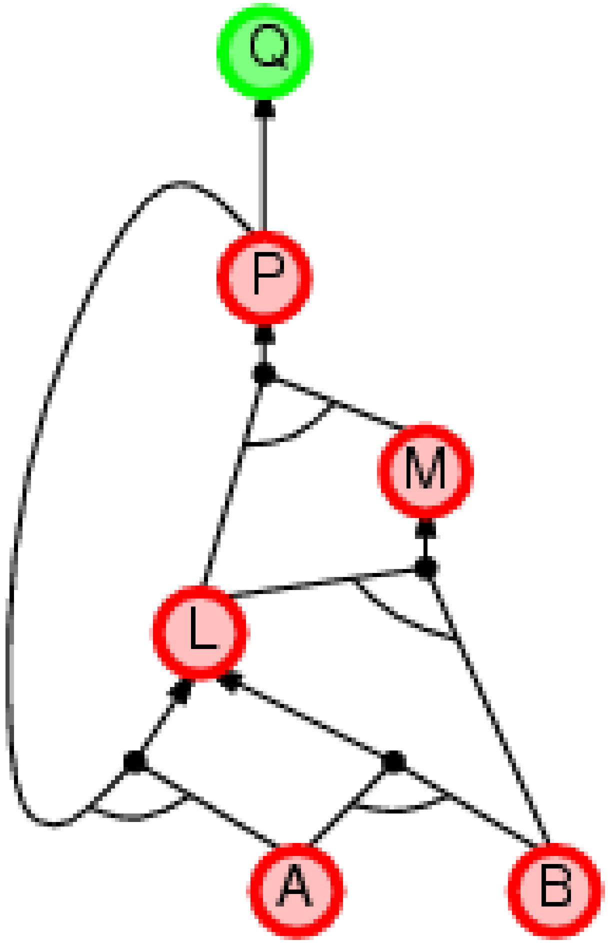
A
B
Q?<=P?
L? ∧ M? => P?
P? ∧ A => L?
A ∧ B => L
L ∧ B => M

Backward chaining example



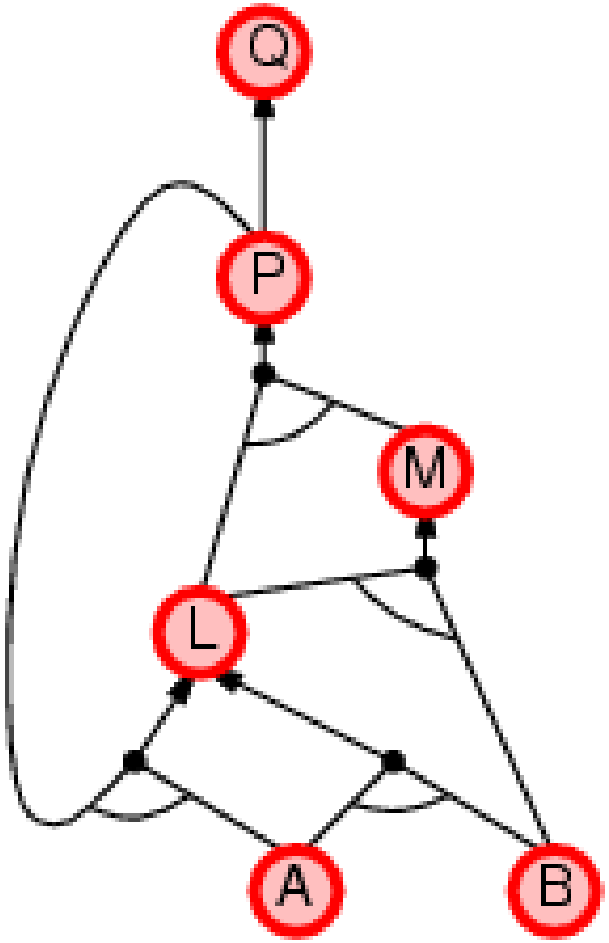
A
B
Q?<=P?
L ∧ *M => P?*
P? ∧ *A => L?*
A ∧ *B => L*
L ∧ *B => M*

Backward chaining example



A
B
Q?<=P
L ∧ *M* => *P*
P? ∧ *A* => *L?*
A ∧ *B* => *L*
L ∧ *B* => *M*

Backward chaining example



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

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

- Look at satisfiability problems in conjunctive normal form
- An underconstrained problem is one with relatively few clause constraining the variables
- Consider random 3-CNF sentences. e.g.,

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

  number of clauses: 5

  number of symbols:5

  This is an easy satisfiability problem. Sixteen of the 32 possible assignments are models of this sentences.

Hard satisfiability problems

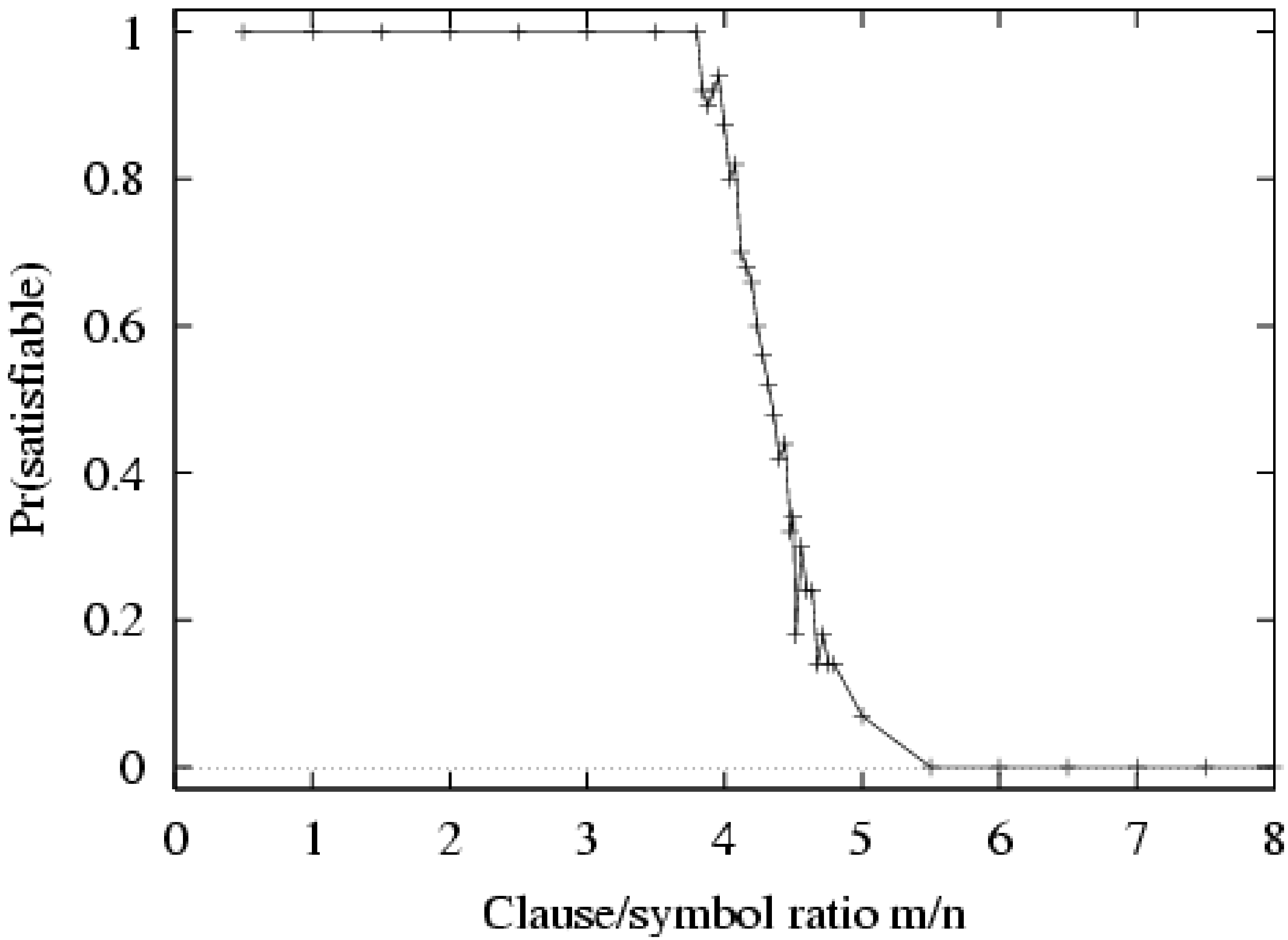
- An overconstrained problem has many clauses relative to the number of variables and is likely to have no solutions.
- The underconstrained problems are the easiest to solve (because it is so easy to guess a solution).

m = number of clauses

n = number of symbols

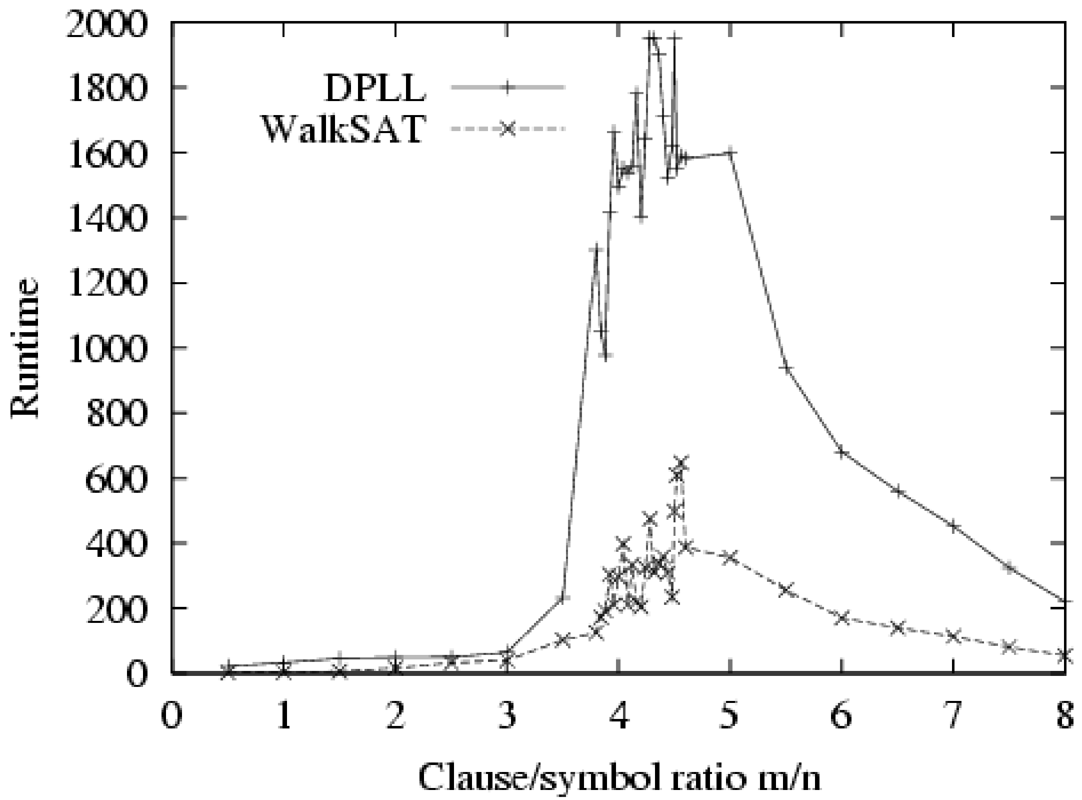
  - Hard problems seem to cluster near $m/n = 4.3$ (critical point)

Hard satisfiability problems



The probability of satisfiability that a random 3-CNF sentence with n=50 symbols

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 $[x^t, y^t]$,

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

- 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

Summary

- 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**

Questions?

作业

- 7.2
- 7.10