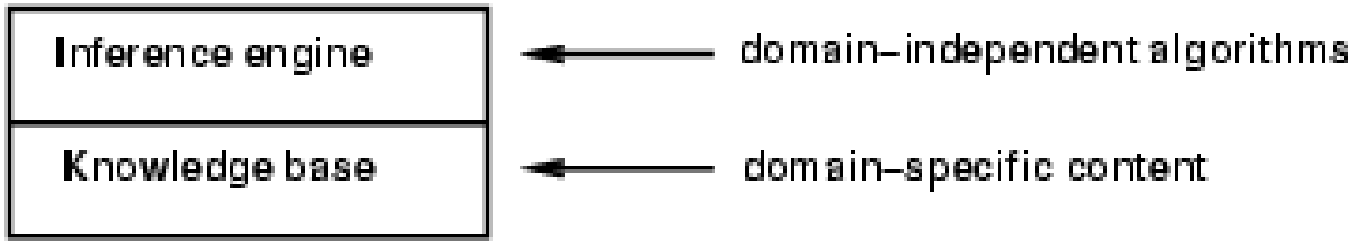
# Propositional Logic

# 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 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

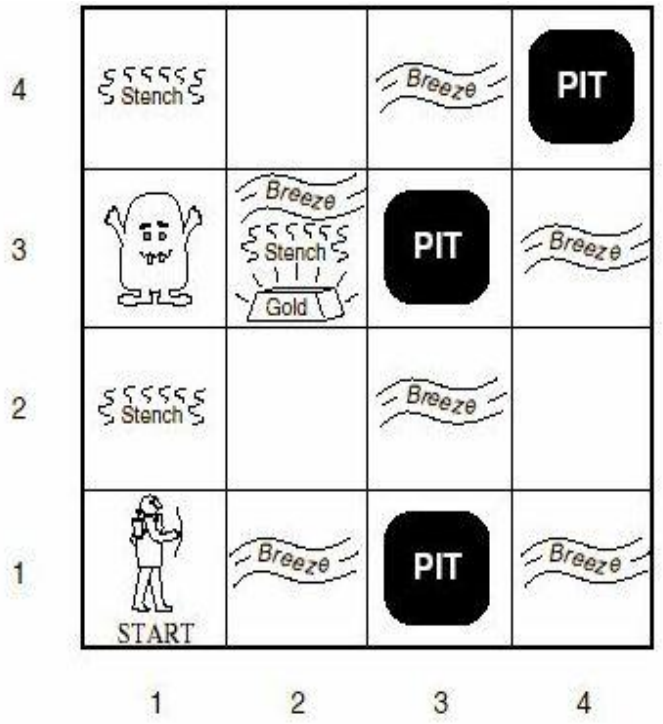
```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    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

# The wumpus world

- A scary world, indeed
  - A maze in a cave
  - A wumpus who will eat you
  - One arrow that can kill the wumpus
  - Pits that can entrap you (but not the wumpus for it is too large to fall in)
  - A heap of gold somewhere



# But you have sensing and action

- Sensing (each is either on or off – a single bit)
  - wumpus emits a stench in adjacent squares
  - pits cause a breeze in adjacent squares
  - gold causes glitter you see when in the square
  - walking into wall causes a bump
  - death of wumpus can be heard everywhere in world

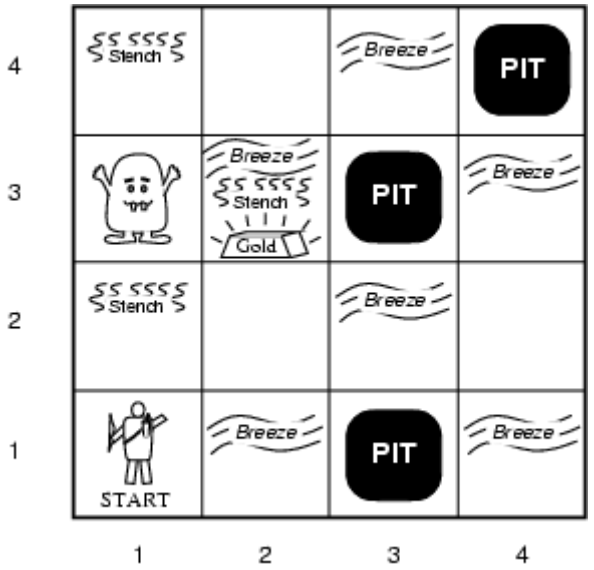
# But you have sensing and action

- Action
  - You can turn left or right 90 degrees
  - You can move forward
  - You can shoot an arrow in your facing direction

# 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

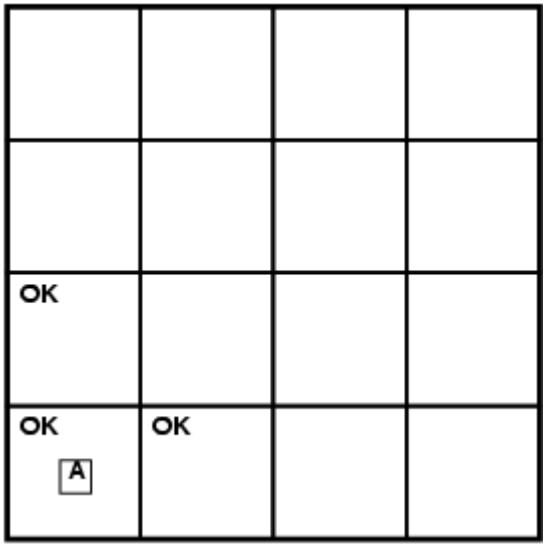



- Sensors: Stench, Breeze, Glitter
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot

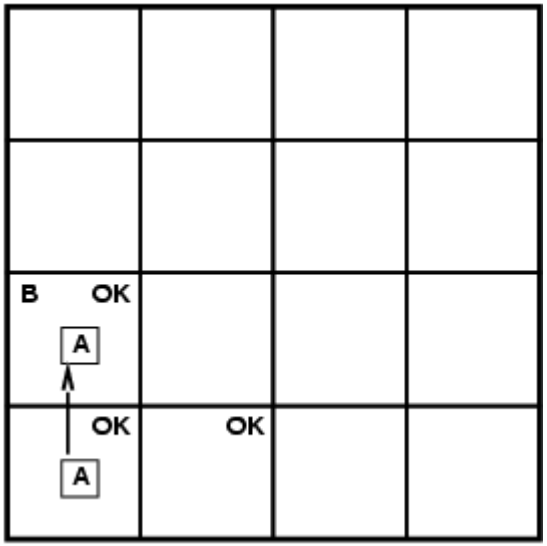
# Wumpus world characterization

- 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

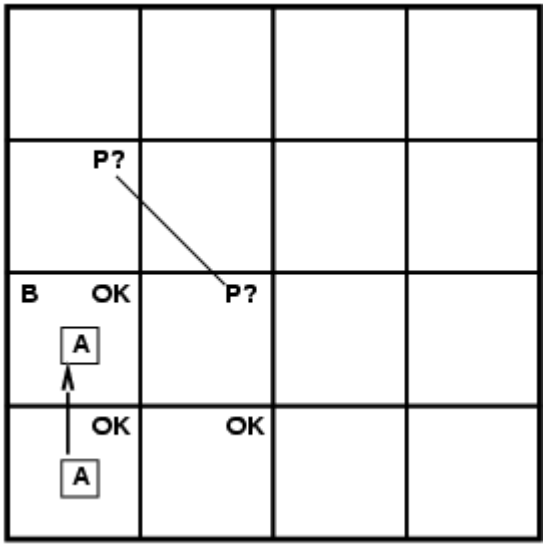
# Exploring a wumpus world



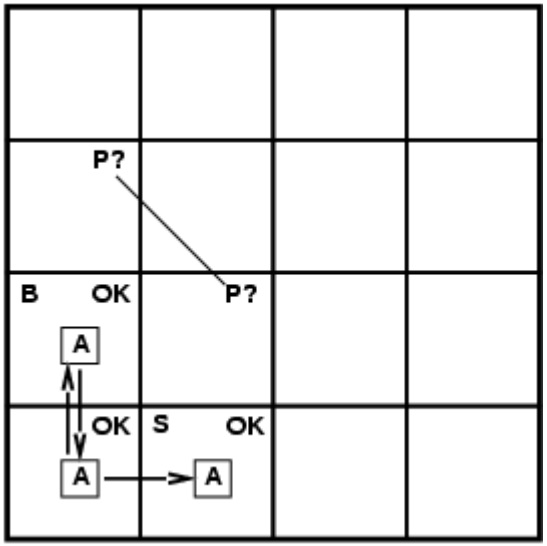
# Exploring a wumpus world



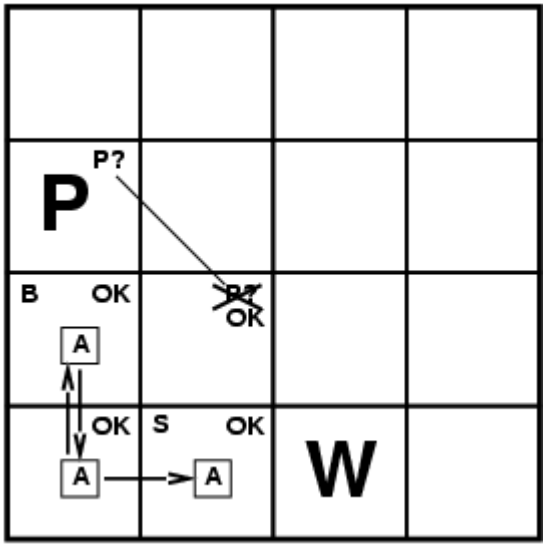
# Exploring a wumpus world



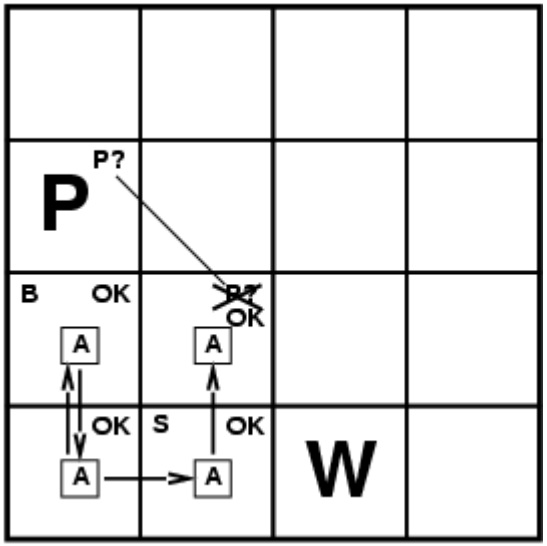
# Exploring a wumpus world



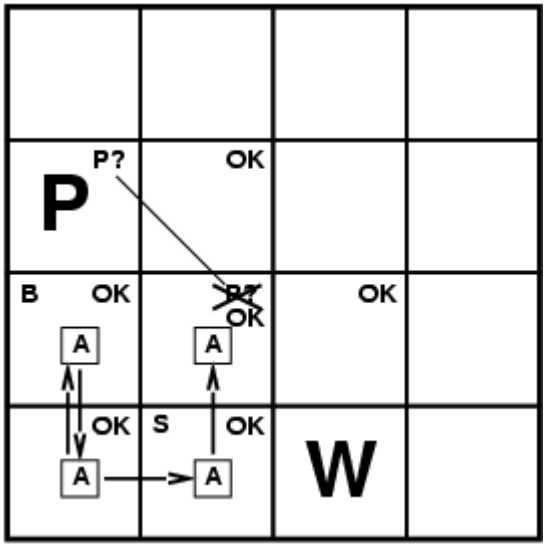
# Exploring a wumpus world



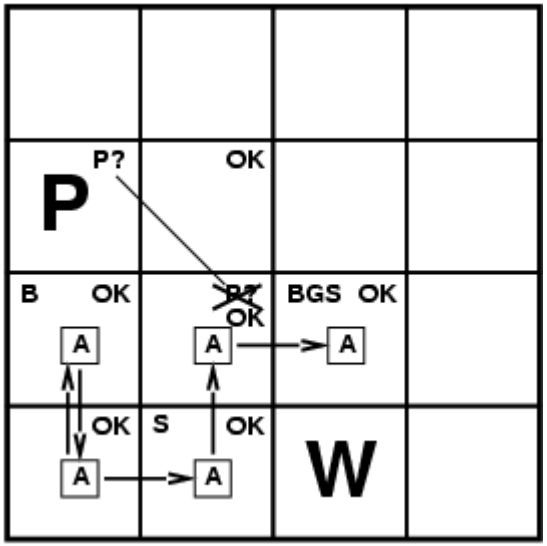
# Exploring a wumpus world



# Exploring a wumpus world



# Exploring a wumpus world



# 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 ≥ y is a sentence; x2+y > {} is not a sentence
  - x+2 ≥ y is true iff the number x+2 is no less than the number y
  - x+2 ≥ y is true in a world where x = 7, y = 1
  - x+2 ≥ y is false in a world where x = 0, y = 6

# Propositional Logic

- **Logical constants**: true, false
- **Propositional symbols**: P, Q, S, ... (**atomic sentences**)
- Wrapping **parentheses**: ( … )
- Sentences are combined by **connectives**:

| | |
|---|---|
| Ù ...and | [conjunction] |
| Ú ...or | [disjunction] |
| Þ ...implies (→, É**)** | [implication / conditional] |
| Û ..is equivalent | [biconditional] |
| Ø ...not | [negation] |

- **Literal**: atomic sentence or negated atomic sentence

# Examples of PL sentences

- $(P \wedge Q) \rightarrow R$

  "If it is hot and humid, then it is raining"

- $Q \rightarrow P$

  "If it is humid, then it is hot"

- Q

  "It is humid."

- A better way:

  Ho = "It is hot"

  Hu = "It is humid"

  R = "It is raining"

# Propositional Logic (PL)

- A simple language useful for showing key ideas and definitions
- User defines a set of propositional symbols, like P and Q.
- User defines the **semantics** of each propositional symbol:
  - P means “It is hot”
  - Q means “It is humid”
  - R means “It is raining”

# Propositional Logic: Syntax

- 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)

# A BNF grammar of sentences in propositional logic

**S := <Sentence> ;**

**<Sentence> := <AtomicSentence> | <ComplexSentence> ;**

**<AtomicSentence> := "TRUE" | "FALSE" |**

**"P" | "Q" | "S" ;**

**<ComplexSentence> := "(" <Sentence> ")" |**

**<Sentence> <Connective> <Sentence> |**

**"NOT" <Sentence> ;**

**<Connective> := "AND" | "OR" | "IMPLIES" | "EQUIVALENT" ;**

# Examples

**(P**Ù**Q)** É**ØP**
**P**É**ØP**
**P**Ú**P**É**P**
**(P**É**Q)** É **(Ø****Q**É**ØP)**
**ØØP**

# Propositional Logic: Semantics

**Each interpretation specifies true/false for each proposition symbol**

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

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

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

| | | | |
|---|---|---|---|
| **ØS** | **is true iff** | **S is false** | |
| **$S_1$ Ù $S_2$** | **is true iff** | **$S_1$ is true and** | **$S_2$ is true** |
| **$S_1$ Ú $S_2$** | **is true iff** | **$S_1$is true or** | **$S_2$ is true** |
| **$S_1$ Þ $S_2$** | **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** |
| **$S_1$ Û $S_2$** | **is true iff** | **$S_1$ Þ $S_2$ is true and** | **$S_2$ Þ $S_1$ is true** |

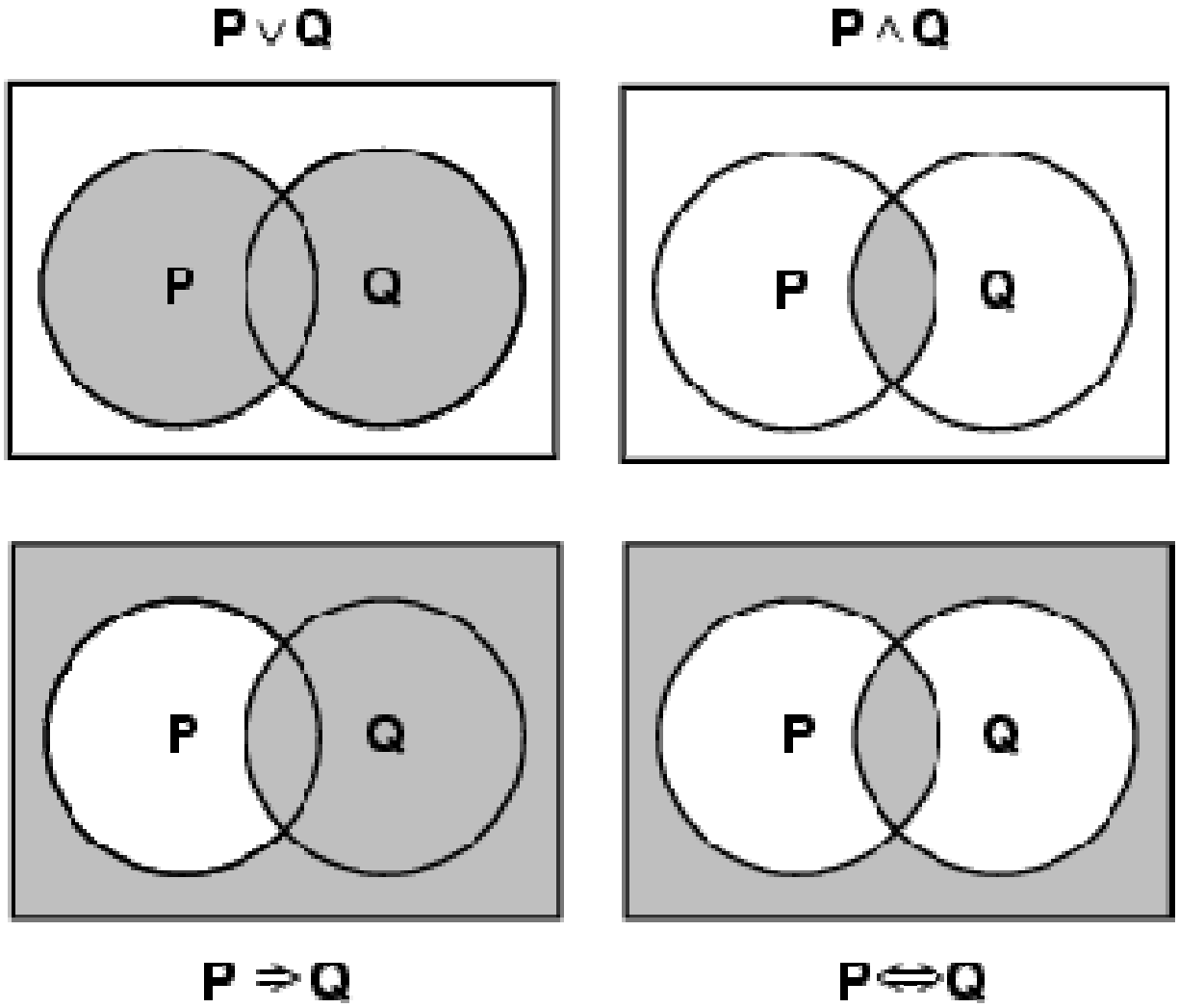
**Simple recursive process evaluates an arbitrary sentence, e.g.,**

$$ØP_{1,2} Ù (P_{2,2} Ú P_{3,1}) = true Ù (true Ú false) = true Ù true = true$$

# Truth tables for connectives

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

# Models



# Truth Tables

**The five logical 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 |

**A complex sentence:**

| $P$ | $H$ | $P \vee H$ | $(P \vee H) \wedge \neg H$ | $((P \vee H) \wedge \neg H) \Rightarrow P$ |
|---|---|---|---|---|
| False | False | False | False | True |
| False | True | True | False | True |
| True | False | True | True | True |
| True | True | True | False | True |

# Some terms

- The meaning or **semantics** of a sentence determines its **interpretation**.
- Given the truth values of all symbols in a sentence, it can be “evaluated” to determine its **truth value** (True or False).
- A **model** for a KB is a “possible world” (assignment of truth values to propositional symbols) in which each sentence in the KB is True.

# More terms

- A **valid sentence** or **tautology** is a sentence that is True under all interpretations, no matter what the world is actually like or what the semantics is. Example: “It’s raining or it’s not raining.”
- An **inconsistent sentence** or **contradiction** is a sentence that is False under all interpretations. The world is never like what it describes, as in “It’s raining and it’s not raining.”

# Entailment

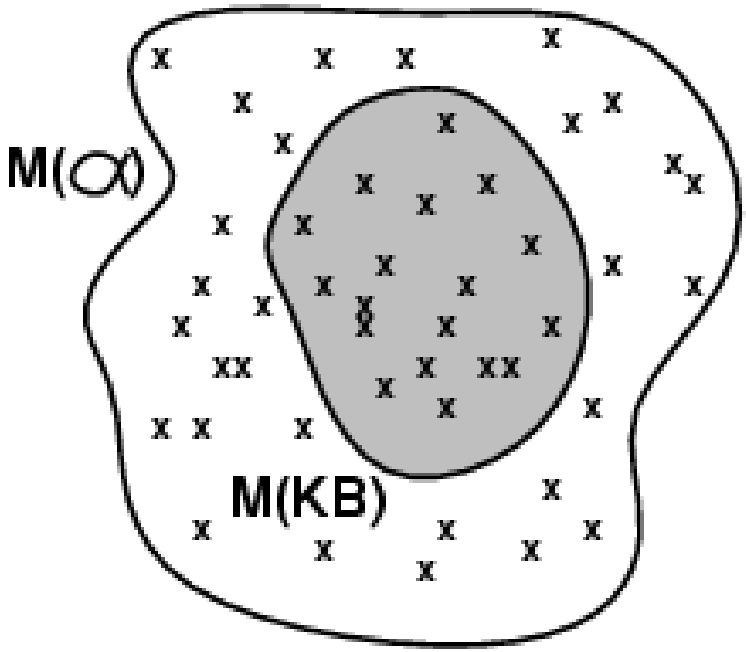
- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- 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 “It is hot” and “It is humid” entails “Either it is hot or it is humid”
  - E.g., x+y = 4 entails  4 = x+y
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

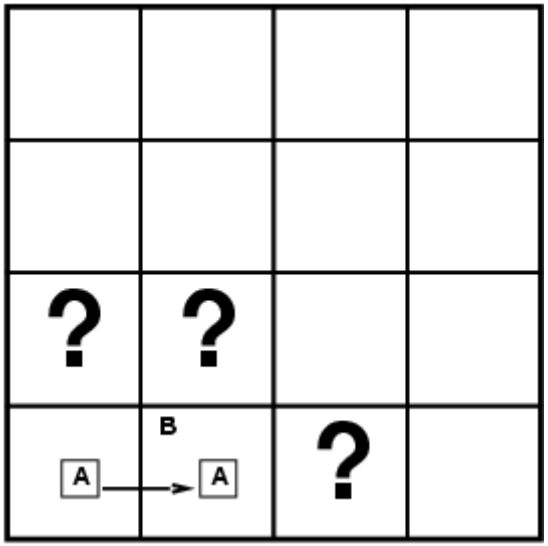
# Models

- 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)* ⊆ *M(*α)



# Entailment in the wumpus world

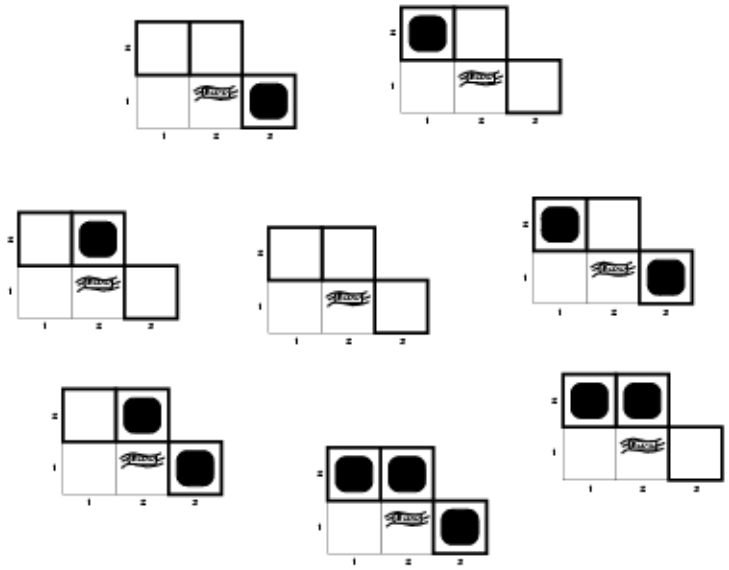
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]



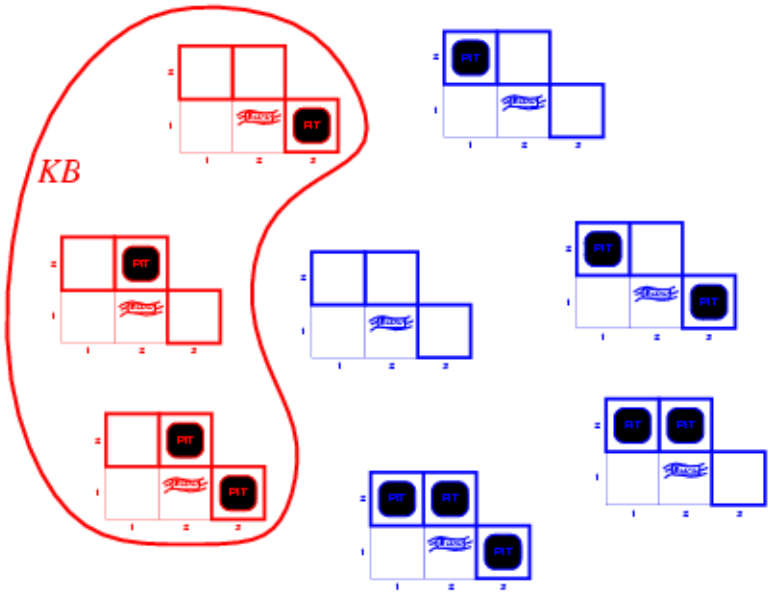

Consider possible models for *KB* assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models

# Wumpus models

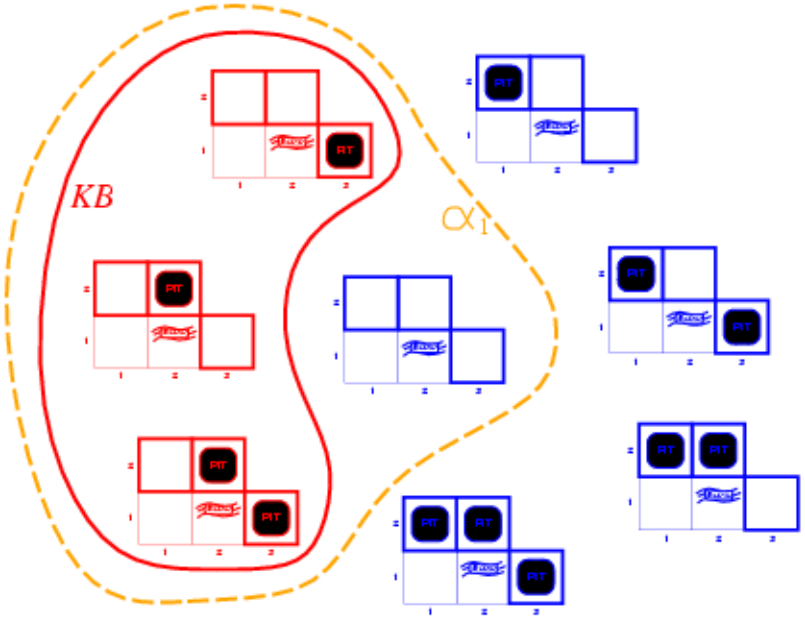


# Wumpus models



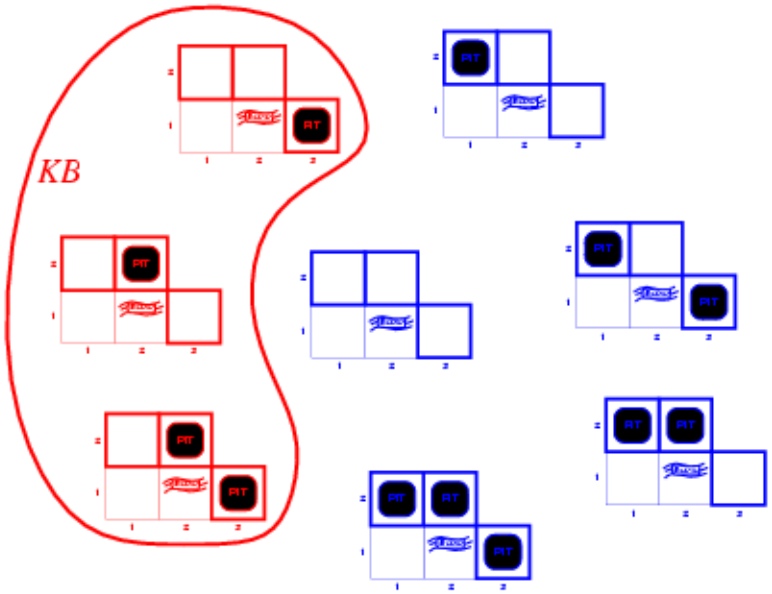

- *KB* = wumpus-world rules + observations

# Wumpus models



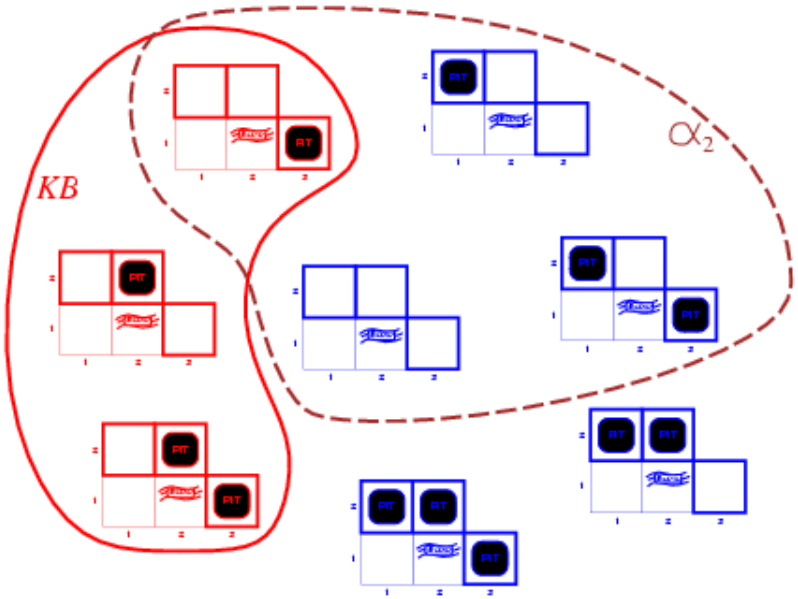

- *KB* = wumpus-world rules + observations
- $\alpha_1$ = "[1,2] is safe", $KB \nvdash \alpha_1$, proved by model checking

# Wumpus models




- *KB* = wumpus-world rules + observations

# Wumpus models




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

# 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"

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

# Combining it all

- 4x4 Wumpus World
  - The “physics” of the game

$$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$$

$$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$$

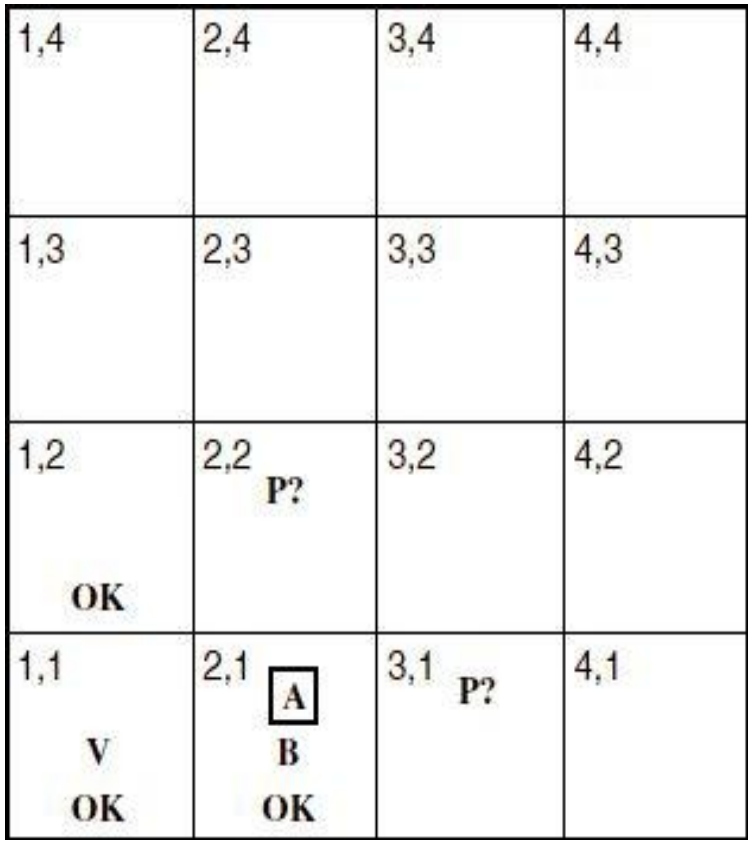
  - A most one wumpus on board (for any two squares, one is free)

$$W_{1,1} \vee W_{1,2} \vee \cdots \vee W_{4,3} \vee W_{4,4}$$

  - Total of 155 sentences containing 64 distinct symbols

# A wumpus knowledge base

- Initial conditions
  - $R_1$: $\sim P_{1,1}$ no pit in [1,1]
- Rules of Breezes (for a few example squares)
  - $R_2$: $B_{1,1}$ **ó** $(P_{1,2} \vee P_{2,1})$
  - $R_3$: $B_{2,1}$ **ó** $(P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- Percepts
  - $R_4$: $\sim B_{1,1}$
  - $R_5$: $B_{2,1}$




- We know: $R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$

# Entailment

- Does KB entail $\alpha$ (KB $\models \alpha$?)
  - Is there a pit in [1,2]: $P_{1,2}$?
  - Consider only what we need
    - $B_{1,1}\, B_{2,1}\, P_{1,1}\, P_{1,2}\, P_{2,1}\, P_{2,2}\, P_{3,1}$
    - $2^7$ permutations of models to check
  - For each model, see if KB is true
  - For all KB = True, see if $\alpha$ is true

**This algorithm is called Model Checking**

# Entailment

- Truth table

| $B_{1,1}$ | $B_{2,1}$ | $P_{1,1}$ | $P_{1,2}$ | $P_{2,1}$ | $P_{2,2}$ | $P_{3,1}$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $KB$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *false* | *false* | *false* | *false* | *false* | *false* | *false* | *true* | *true* | *true* | *true* | *false* | *false* |
| *false* | *false* | *false* | *false* | *false* | *false* | *true* | *true* | *true* | *false* | *true* | *false* | *false* |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| *false* | *true* | *false* | *false* | *false* | *false* | *false* | *true* | *true* | *false* | *true* | *true* | *false* |
| *false* | *true* | *false* | *false* | *false* | *false* | *true* | *true* | *true* | *true* | *true* | *true* | *true* |
| *false* | *true* | *false* | *false* | *false* | *true* | *false* | *true* | *true* | *true* | *true* | *true* | *true* |
| *false* | *true* | *false* | *false* | *false* | *true* | *true* | *true* | *true* | *true* | *true* | *true* | *true* |
| *false* | *true* | *false* | *false* | *true* | *false* | *false* | *true* | *false* | *false* | *true* | *true* | *false* |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| *true* | *true* | *true* | *true* | *true* | *true* | *true* | *false* | *true* | *true* | *false* | *true* | *false* |

**There is no pit in $P_{1,2}$**

# Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from *KB* by procedure *i*
- Soundness: *i* is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
- Completeness: *i* is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

# Inference rules

- **Logical inference** is used to create new sentences that logically follow from a given set of sentences (KB).

- An inference rule is **sound** if every sentence X produced by an inference rule operating on a KB logically follows from the KB. (That is, the inference rule does not create any contradictions)

- An inference rule is **complete** if it is able to produce every expression that logically follows from (is entailed by) the KB.

# Inference rules

- Inference Rules $\dfrac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$
  - Modus Ponens:
    - Whenever sentences of form $\alpha$ => $\beta$ and $\alpha$ are given
      the sentence $\beta$ can be inferred
      - $R_1$: Green => Martian
      - $R_2$: Green
      - Inferred: Martian

# Inference rules

- Inference Rules
  - And-Elimination
    - Any of conjuncts can be inferred

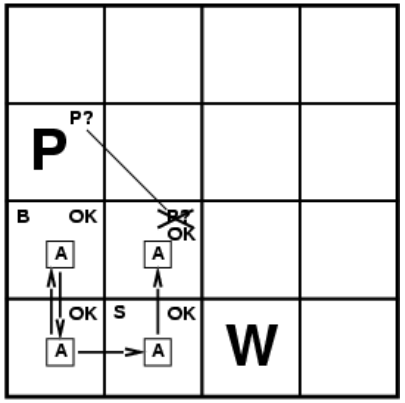
$$\frac{\alpha \wedge \beta}{\alpha}$$

      - $R_1$: Martian ^ Green
      - Inferred: Martian
      - Inferred: Green
- Use truth tables if you want to confirm inference rules

# Inference Rule: Resolution

- Conjunctive Normal Form (CNF)
  conjunction of disjunctions of clauses

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




- 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., $\dfrac{P_{1,3} \vee P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$

- Resolution is sound but not complete

# Inference Rule: Resolution

Chaining

$$\frac{P \Rightarrow Q,\ Q \Rightarrow R}{P \Rightarrow R}$$

Modus Ponens

$$\frac{P,\ P \Rightarrow Q}{Q}$$

# Resolution is sound

Soundness of resolution inference rule:

$$\frac{S_1 \cup \{l\} \qquad S_1 \cup \{\neg l\}}{S_1 \cup S_2}$$

# Resolution is not complete

Not completeness of resolution inference rule:

PÙR |= PÚR

But we cannot use resolution to prove P∨R from P∧R

# Sound rules of inference

- Here are some examples of sound rules of inference
  - *A rule is sound if its conclusion is true whenever the premise is true*
- Each can be shown to be sound using a truth table

| RULE | PREMISE | CONCLUSION |
|---|---|---|
| Modus Ponens | A, A $\rightarrow$ B | B |
| And Introduction | A, B | A $\wedge$ B |
| And Elimination | A $\wedge$ B | A |
| Double Negation | $\neg\neg$A | A |
| Unit Resolution | A $\vee$ B, $\neg$B | A |
| **Resolution** | **A Ú B, ØB Ú C** | **A Ú C** |

# Soundness of modus ponens

| **A** | **B** | **A → B** | **OK?** |
|---|---|---|---|
| True | True | True | Ö |
| True | False | False | Ö |
| False | True | True | Ö |
| False | False | True | Ö |

# Soundness of the resolution inference rule

| $\alpha$ | $\beta$ | $\gamma$ | $\alpha \vee \beta$ | $\neg\beta \vee \gamma$ | $\alpha \vee \gamma$ |
|---|---|---|---|---|---|
| False | False | False | False | True | False |
| False | False | True | False | True | True |
| False | True | False | True | False | False |
| False | True | True | True | True | True |
| True | False | False | True | True | True |
| True | False | True | True | True | True |
| True | True | False | True | False | True |
| True | True | True | True | True | True |

# Logical equivalence

- Two sentences are logically equivalent iff true in same models: α ≡ ß iff α ╞ β and β ╞ α

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

# Proving things

- A **proof** is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference.
- The last sentence is the **theorem** (also called goal or query) that we want to prove.
- Example:

| | | |
|---|---|---|
| 1 Hu | Premise | "It is humid" |
| 2 Hu$\rightarrow$Ho | Premise | "If it is humid, it is hot" |
| 3 Ho | Modus Ponens(1,2) | "It is hot" |
| 4 (Ho$\wedge$Hu)$\rightarrow$R | Premise | "If it's hot & humid, it's raining" |
| 5 Ho$\wedge$Hu | And Introduction(1,2) | "It is hot and humid" |
| 6 R | Modus Ponens(4,5) | "It is raining" |

# The “Hunt the Wumpus” agent

- Some atomic propositions:

  S12 = There is a stench in cell (1,2)
  B34 = There is a breeze in cell (3,4)
  W22 = The Wumpus is in cell (2,2)
  V11 = We have visited cell (1,1)
  OK11 = Cell (1,1) is safe.
  etc

- Some rules:

  $\neg S11 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W21$
  $\neg S21 \rightarrow \neg W11 \wedge \neg W21 \wedge \neg W22 \wedge \neg W31$
  $\neg S12 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W22 \wedge \neg W13$
  $S12 \rightarrow W13 \vee W12 \vee W22 \vee W11$
  etc

- Note that the lack of variables requires us to give similar rules for each cell

# Example of a proof of $\sim P_{1,2}$

- There is no pit in [1,1]:

$$R_1 : \quad \neg P_{1,1} .$$

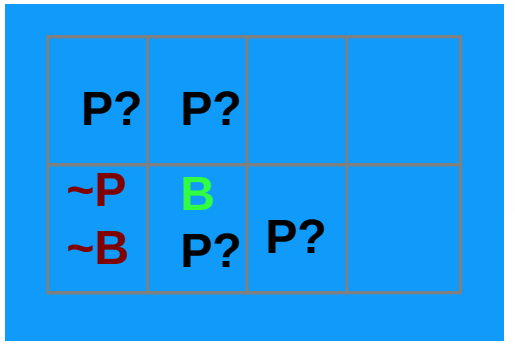
- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:

$$R_2 : \quad B_{1,1} \quad \Leftrightarrow \quad (P_{1,2} \vee P_{2,1}) .$$
$$R_3 : \quad B_{2,1} \quad \Leftrightarrow \quad (P_{1,1} \vee P_{2,2} \vee P_{3,1}) .$$

- The preceding sentences are true in all wumpus worlds. Now we include the breeze percepts for the first two squares visited in the specific world the agent is in, leading up to the situation in Figure

$$R_4 : \quad \neg B_{1,1} .$$
$$R_5 : \quad B_{2,1} .$$

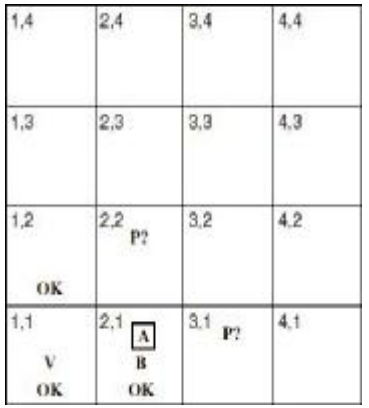
| | | | |
|---|---|---|---|
| 1,4 | 2,4 | 3,4 | 4,4 |
| 1,3 | 2,3 | 3,3 | 4,3 |
| 1,2 OK | 2,2 P? | 3,2 | 4,2 |
| 1,1 V OK | 2,1 A B OK | 3,1 P? | 4,1 |



# Example of a proof of ~$P_{1,2}$

that is, there is no pit in [1,2]. First, we apply biconditional elimination to $R_2$ to obtain

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




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

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

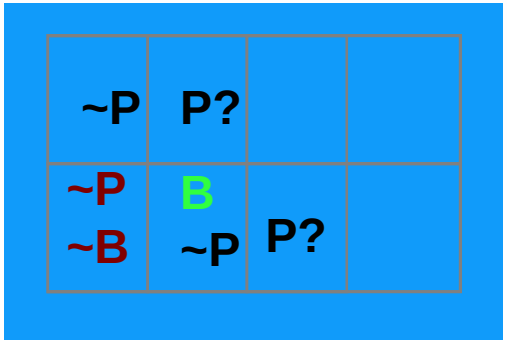
Now we can apply Modus Ponens with $R_8$ and the percept $R_4$ (i.e., $\neg B_{1,1}$), to obtain

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

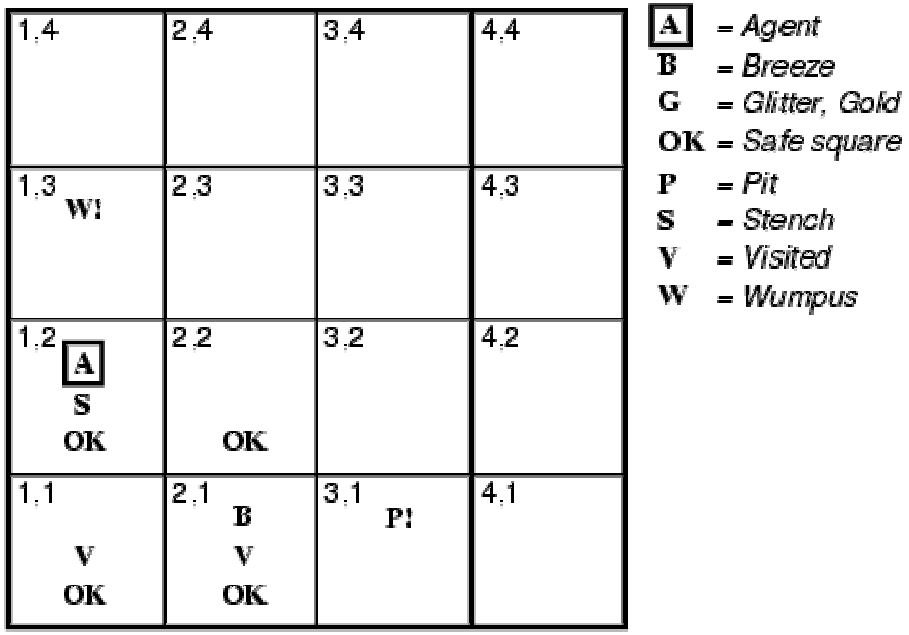
Finally, we apply de Morgan's rule, giving the conclusion

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

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



# After the third move




- We know the following facts:

  ¬S11

  ¬S21

  S12

  ¬W11

  ¬W22

  . . .

- We can prove that the Wumpus is in (1,3) using the four rules:

  (R1) $\neg S11 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W21$
  (R2) $\neg S21 \rightarrow \neg W11 \wedge \neg W21 \wedge \neg W22 \wedge \neg W31$
  (R3) $\neg S12 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W22 \wedge \neg W13$
  (R4) $S12 \rightarrow W13 \vee W12 \vee W22 \vee W11$

# Proving W13

- Apply MP with $\neg$S11 and R1 :
  $\neg$ W11 $\wedge$ $\neg$ W12 $\wedge$ $\neg$ W21
- Apply And-Elimination to this, yielding 3 sentences:
  $\neg$ W11, $\neg$ W12, $\neg$ W21
- Apply MP to $\neg$S21 and R2, then apply And-elimination:
  $\neg$ W22, $\neg$ W21, $\neg$ W31
- Apply MP to S12 and R4 to obtain:
  W13 $\vee$ W12 $\vee$ W22 $\vee$ W11
- Apply Unit resolution on (W13 $\vee$ W12 $\vee$ W22 $\vee$ W11) and $\neg$W11:
  W13 $\vee$ W12 $\vee$ W22
- Apply Unit Resolution with (W13 $\vee$ W12 $\vee$ W22) and $\neg$W22:
  W13 $\vee$ W12
- Apply Resolution with (W13 $\vee$ W12) and $\neg$W12:
  W13

# Problems with the propositional Wumpus hunter

- Lack of variables prevents stating more general rules
  - We need a set of similar rules for each cell

# Conversion to CNF

**$B_{1,1}$ Û $(P_{1,2}$ Ú $P_{2,1})$**

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

# Conversion to CNF

Ø(PÉQ)Ú(RÉP)

1. Ø(Ø PÚQ)Ú( ØRÚP)
2. (PÙØQ)Ú( ØRÚP)
3. (PÚØRÚP) Ù ( ØQ ÚØRÚP)
4. (PÚØR) Ù( ØQ ÚØRÚP)

# Resolution Refutations 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 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 Refutations is sound and complete**

# Example

**BAT_OK**
**LIFTABLE**
**MOVES**

**BAT_OK Ù LIFTABLE É MOVES**

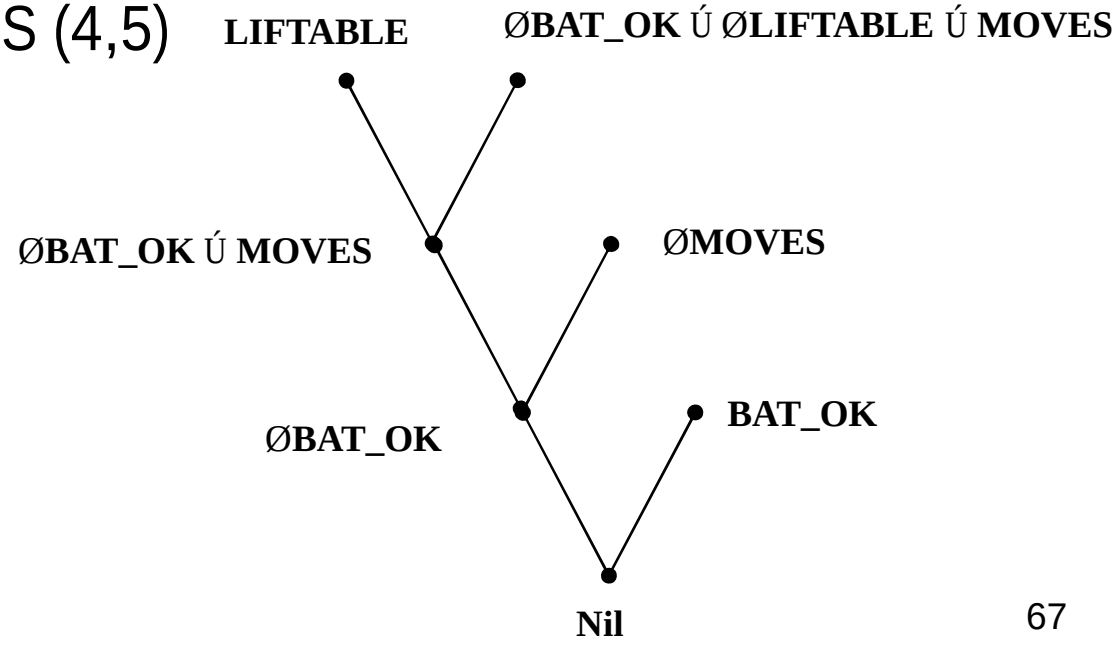
The robot can sense BAT_OK and MOVES but not LIFTABLE.

# Resolution Refutation example

- *KB* =

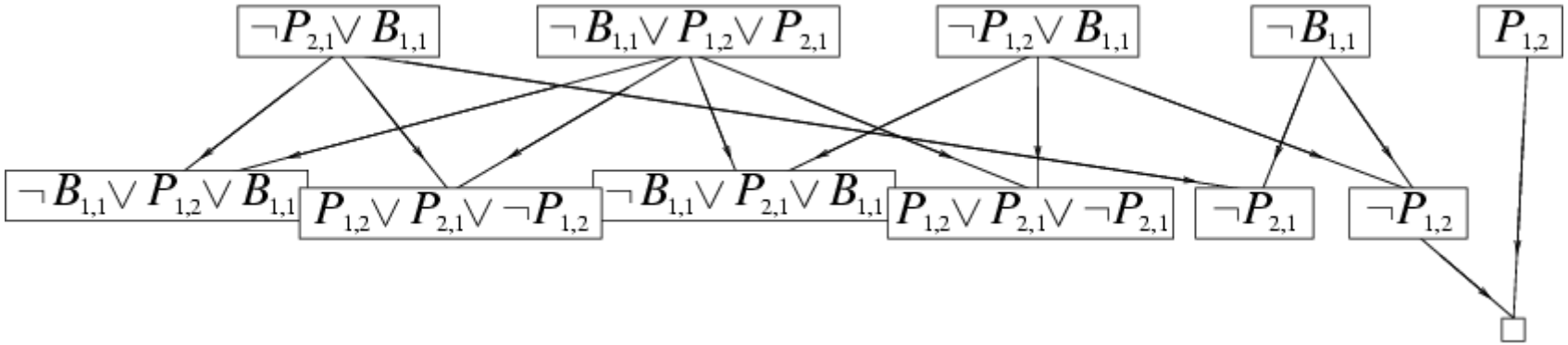
  1.BAT_OK
  2.ØMOVES
  3.BAT_OK Ù LIFTABLE É MOVES
- **α** = ØLIFTABLE

4. ØBAT_OK Ú ØLIFTABLE Ú MOVES (CNF of 3.)
5. LIFTABLE (negation of theorem **α**)
6. ØBAT_OK Ú MOVES (4,5)
7. ØBAT_OK (6,2)
8. Nil (7,1)



# Resolution Refutation example

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



# Resolution search strategies

- There are a number of general (domain-independent) strategies that are useful in controlling a resolution theorem prover
- We'll briefly look at the following:
  - Breadth-first
  - Length heuristics
  - Set of support

# Example

1. ØBattery-OK Ú ØBulbs-OK Ú Headlights-Work
2. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank Ú Engine-Starts
3. ØEngine-Starts Ú Flat-Tire Ú Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ØEmpty-Gas-Tank
8. ØCar-OK
9. ØFlat-Tire  negated goal

# Breadth-first search

- Level 0 clauses are the original axioms and the negation of the goal
- Level k clauses are the resolvents computed from two clauses, one of which must be from level k-1 and the other from any earlier level
- Compute all possible level 1 clauses, then all possible level 2 clauses, etc.
- Complete, but very inefficient

# BFS example

1. ØBattery-OK Ú ØBulbs-OK Ú Headlights-Work
2. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank Ú Engine-Starts
3. ØEngine-Starts Ú Flat-Tire Ú Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ØEmpty-Gas-Tank
8. ØCar-OK
9. ØFlat-Tire

1,4 10. ØBattery-OK Ú ØBulbs-OK
1,5 11. ØBulbs-OK Ú Headlights-Work
2,3 12. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank Ú Flat-Tire Ú Car-OK
2,5 13. ØStarter-OK Ú Empty-Gas-Tank Ú Engine-Starts
2,6 14. ØBattery-OK Ú Empty-Gas-Tank Ú Engine-Starts
2,7 15. ØBattery-OK Ø Starter-OK Ú Engine-Starts
16. … [and we’re still only at Level 1!]

# Length heuristics

- **Shortest-clause heuristic**:
  Generate a clause with the fewest literals first
- **Unit resolution**:
  Prefer resolution steps in which at least one parent clause is a “unit clause,” i.e., a clause containing a single literal
  - Not complete in general, but complete for Horn clause KBs

# Unit resolution example

1. ØBattery-OK Ú ØBulbs-OK Ú Headlights-Work
2. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank Ú Engine-Starts
3. ØEngine-Starts Ú Flat-Tire Ú Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ØEmpty-Gas-Tank
8. ØCar-OK
9. ØFlat-Tire
10. ØBulbs-OK Ú Headlights-Work

1,5

11. ØStarter-OK Ú Empty-Gas-Tank Ú Engine-Starts

2,5

12. ØBattery-OK Ú Empty-Gas-Tank Ú Engine-Starts

2,6

13. ØBattery-OK Ø Starter-OK Ú Engine-Starts

2,7

14. ØEngine-Starts Ú Flat-Tire

3,8

15. ØEngine-Starts Ø Car-OK

3,9

16. … [this doesn't seem to be headed anywhere either!]

# Set of support

- At least one parent clause must be the negation of the goal *or* a “descendant” of such a goal clause (i.e., derived from a goal clause)
- *(When there’s a choice, take the most recent descendant)*
- Complete (assuming all possible set-of-support clauses are derived)
- Gives a goal-directed character to the search

# Set of support example

1. ØBattery-OK Ú ØBulbs-OK Ú Headlights-Work
2. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank Ú Engine-Starts
3. ØEngine-Starts Ú Flat-Tire Ú Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ØEmpty-Gas-Tank
8. ØCar-OK
9. ØFlat-Tire

9,3 10. ØEngine-Starts Ú Car-OK

10,2 11. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank Ú Car-OK

10,8 12. ØEngine-Starts

11,5 13. ØStarter-OK Ú Empty-Gas-Tank Ú Car-OK

11,6 14. ØBattery-OK Ú Empty-Gas-Tank Ú Car-OK

11,7 15. ØBattery-OK Ú ØStarter-OK Ú Car-OK

16. … [a bit more focused, but we still seem to be wandering]

# Unit resolution + set of support example

1. ØBattery-OK Ú ØBulbs-OK Ú Headlights-Work
2. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank Ú Engine-Starts
3. ØEngine-Starts Ú Flat-Tire Ú Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ØEmpty-Gas-Tank
8. ØCar-OK
9. ØFlat-Tire

9,3 10. ØEngine-Starts Ú Car-OK

10,8 11. ØEngine-Starts

12,2 12. ØBattery-OK Ú ØStarter-OK Ú Empty-Gas-Tank

12,5 13. ØStarter-OK Ú Empty-Gas-Tank

13,6 14. Empty-Gas-Tank

14,7 15. FALSE

77 [Hooray! Now that's more like it!]

# Example

$P \vee Q$

$P \vee \neg Q$

$\neg P \vee Q$

Prove $(P \wedge Q)$.

# Horn sentences

- A **Horn sentence** or **Horn clause** has the form:

  $P1 \wedge P2 \wedge P3 \ldots \wedge Pn \rightarrow Q$

or alternatively

  $\neg P1 \vee \neg P2 \vee \neg P3 \ldots \vee \neg Pn \vee Q$

***(P ® Q) = (ØP Ú Q)***

where Ps and Q are non-negated atoms

- To get a proof for Horn sentences, apply Modus Ponens repeatedly until nothing can be done
- We will use the Horn clause form later

# What's good about Horn Clauses

- Two new inference algorithms are possible
  - Forward chaining
  - Backward chaining
- Entailment computation is linear in size of KB

# Forward Chaining

- Does KB (Horn clauses) entail q (single symbol)?
  - Keep track of known facts (positive literals)
  - If the premises of an implication are known
    - Add the conclusion to the known set of facts
  - Repeat process until no further inferences or q is added

# Forward Chaining

- Properties
  - Sound
  - Complete
    - All entailed atomic sentences will be derived
- Data Driven
  - Start with what we know
  - Derive new info until we discover what we want

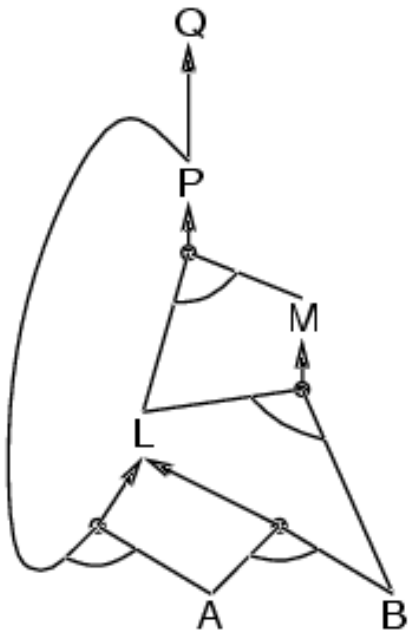
# Backward Chaining

- Start with what you want to know, a query (q)
  - Look for implications that conclude q
    - Look at the premises of those implications
      - Look for implications that conclude those premises…
- Goal-Directed Reasoning
  - Can be less complex search than forward chaining

# Forward chaining

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

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



# 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 and backward chaining

- Horn Form (restricted)

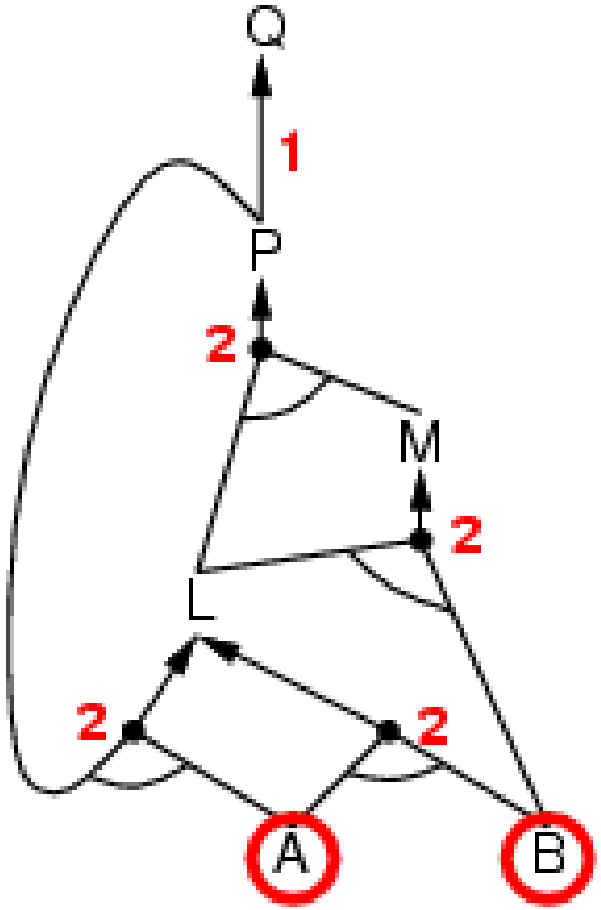
  KB = conjunction of Horn clauses

  - 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

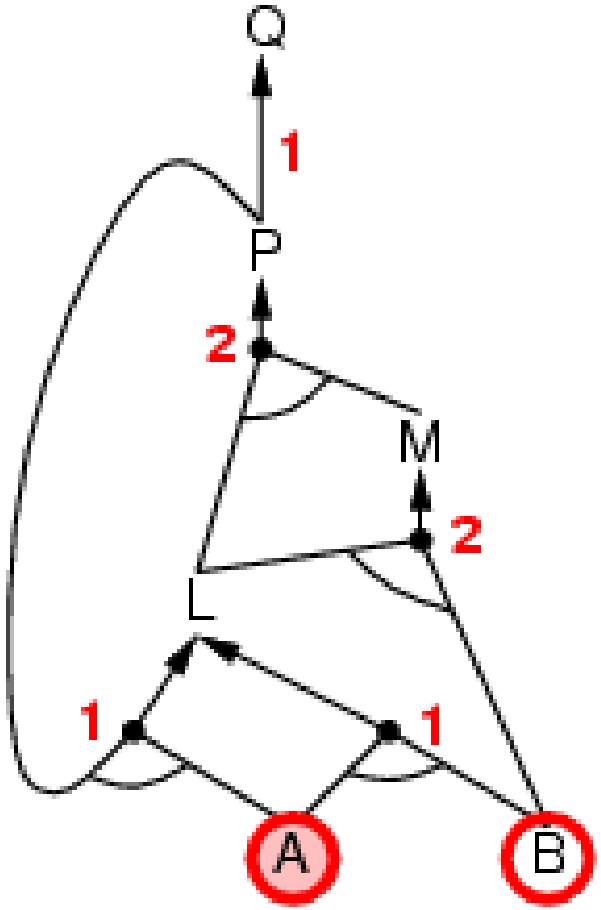
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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

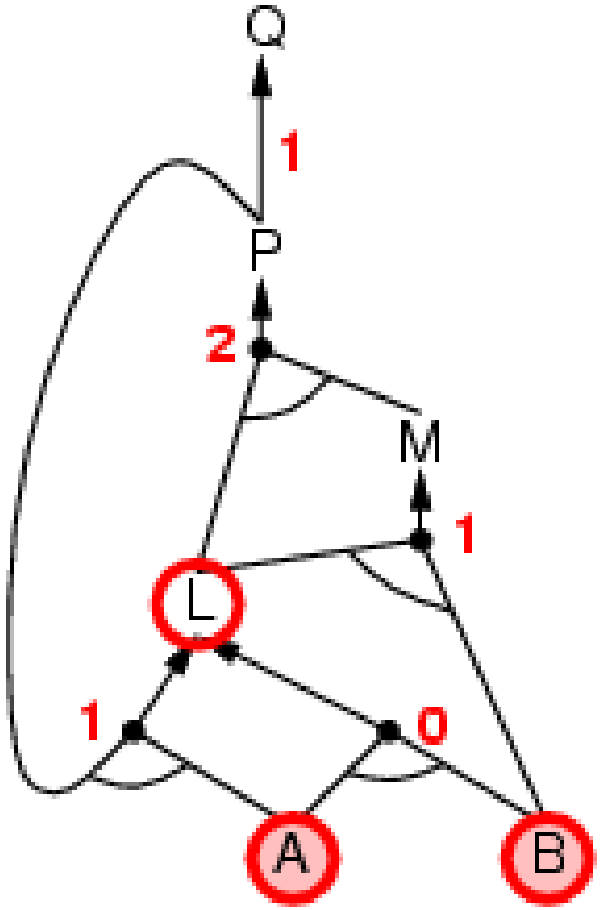
# Forward chaining example



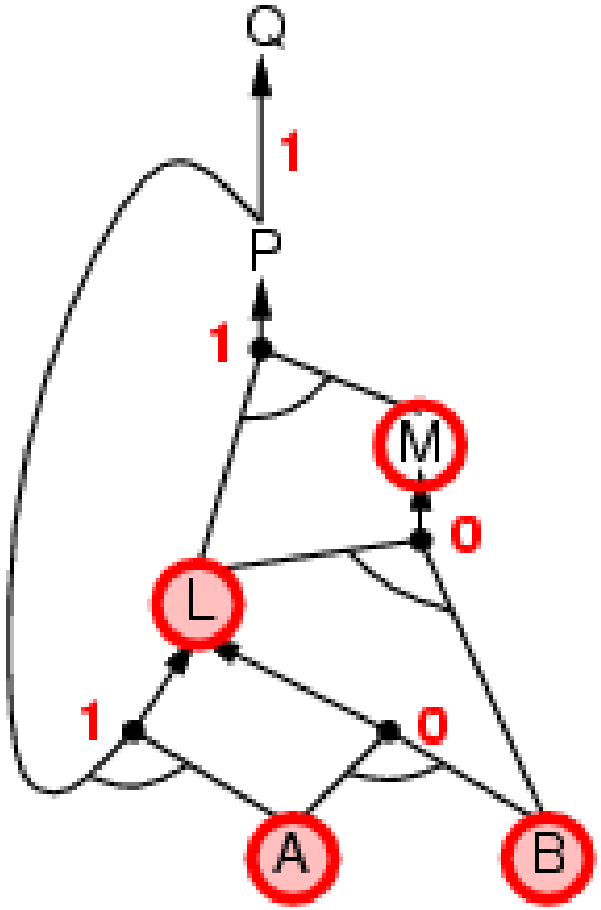
# Forward chaining example



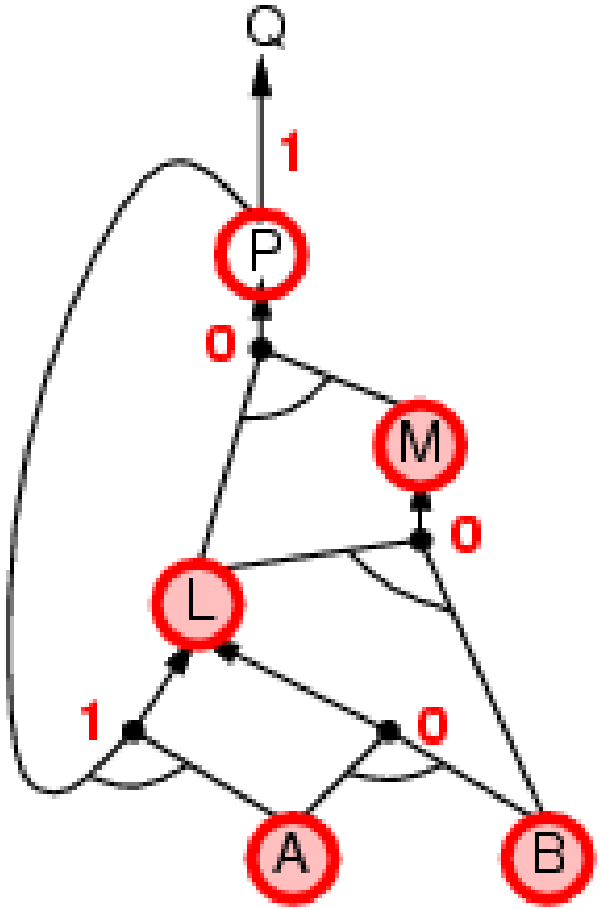
# Forward chaining example



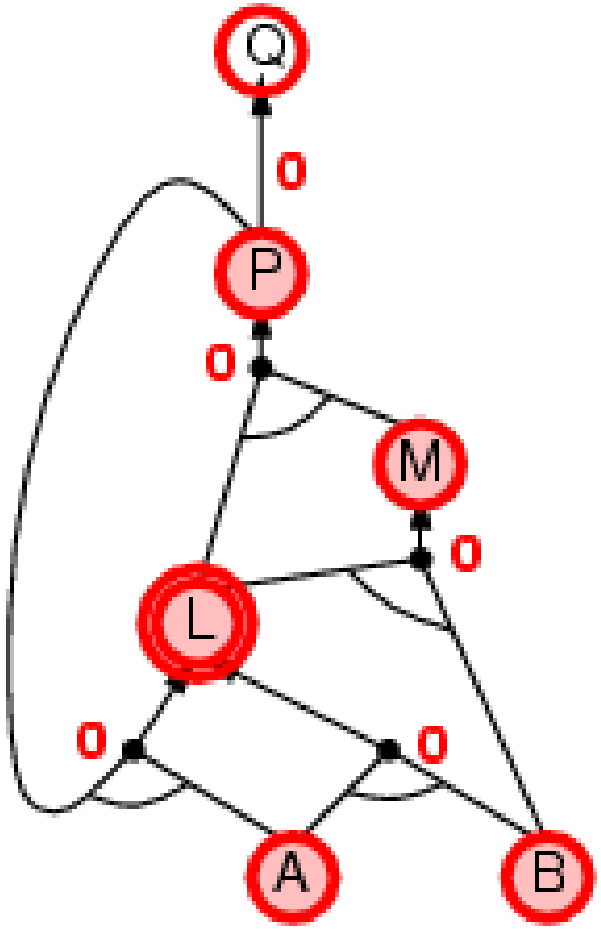
# Forward chaining example



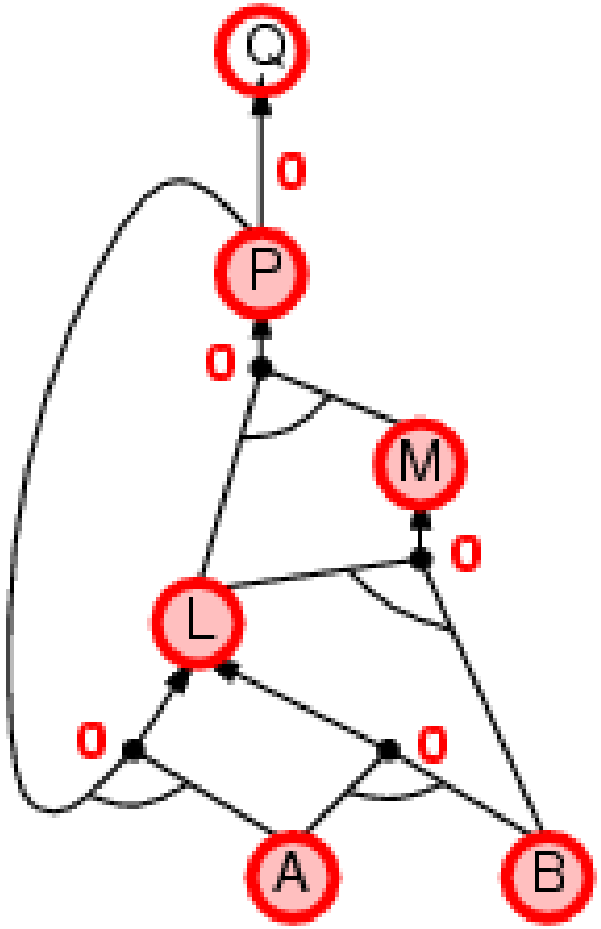
# Forward chaining example



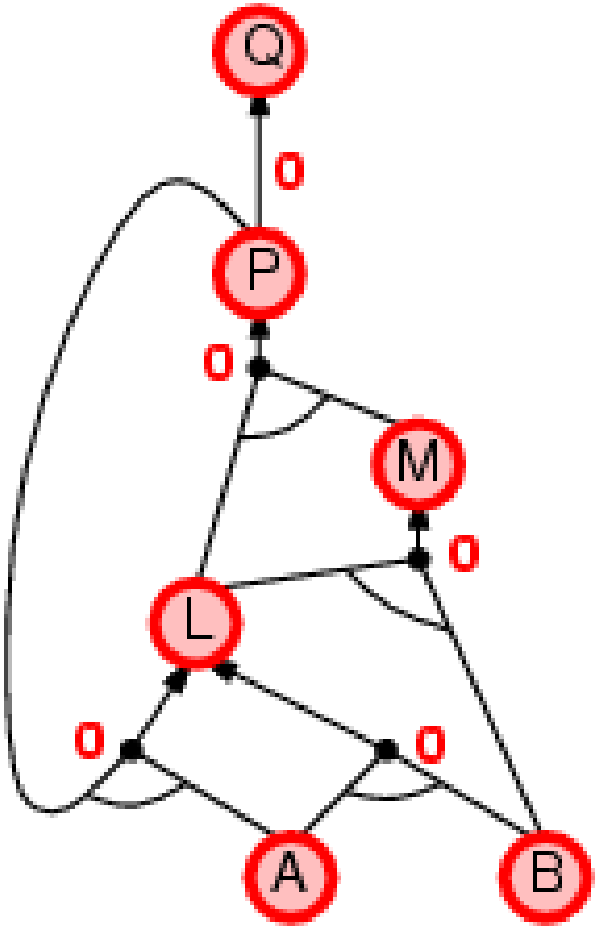
# Forward chaining example



# 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

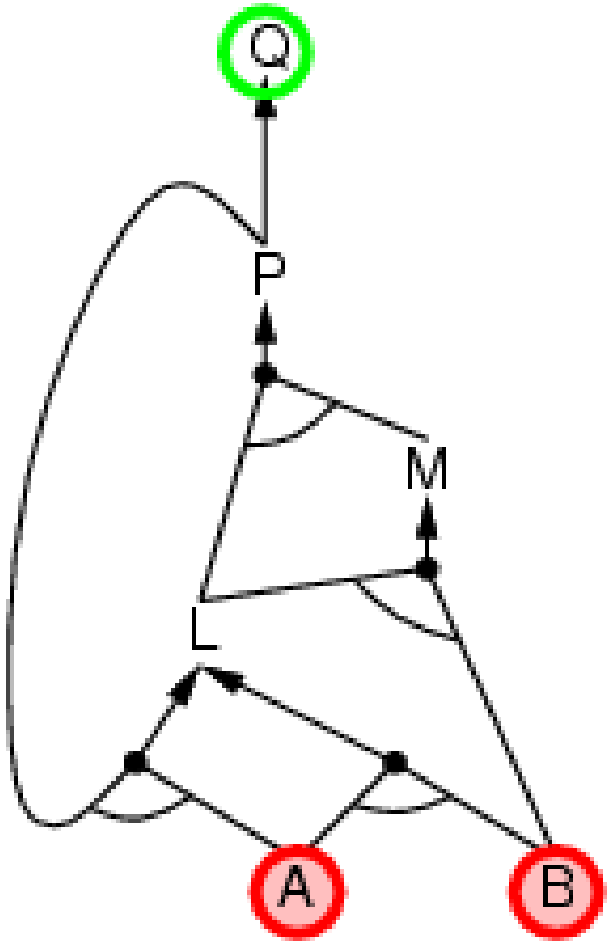
prove by BC all premises of some rule concluding *q*

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

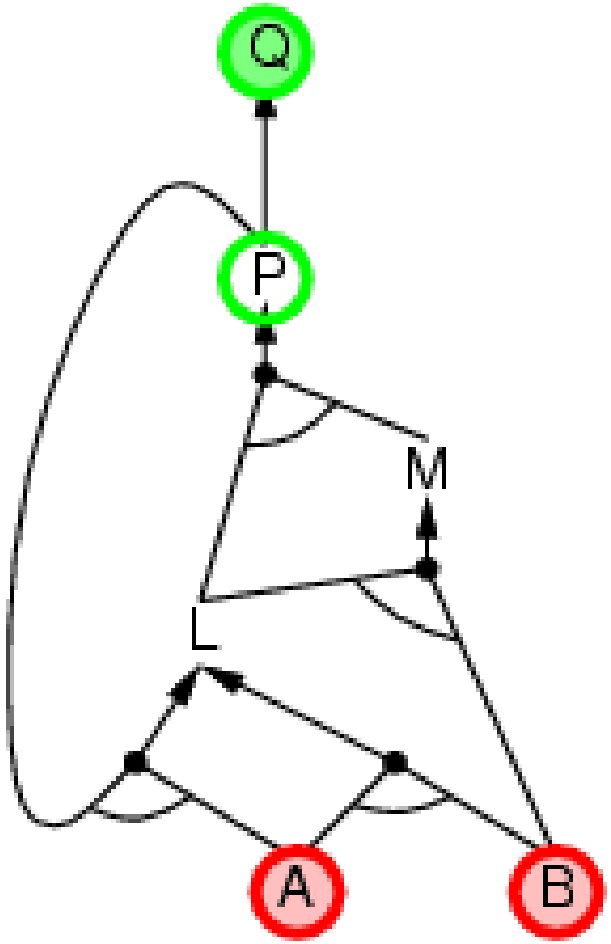
Avoid repeated work: check if new subgoal

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

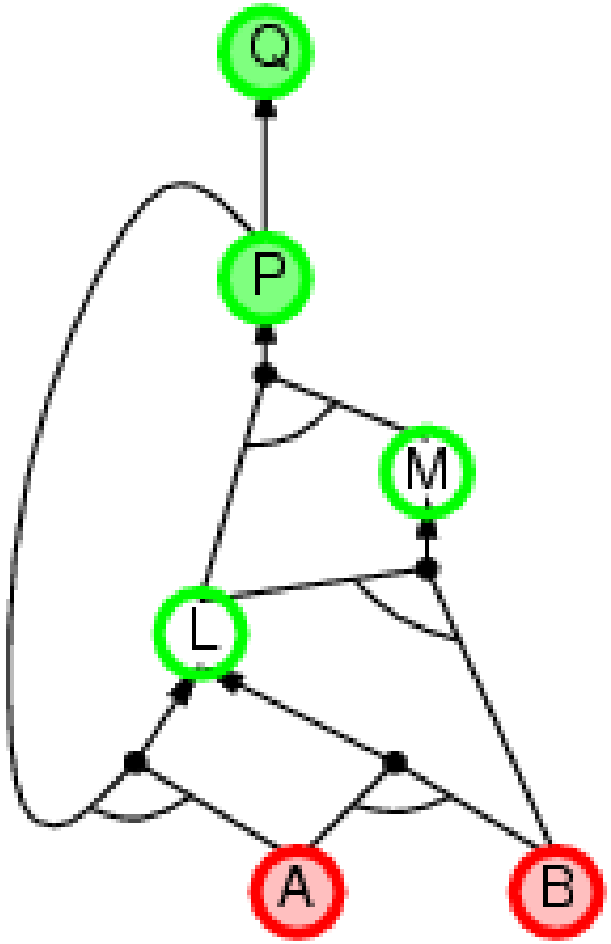
# Backward chaining example



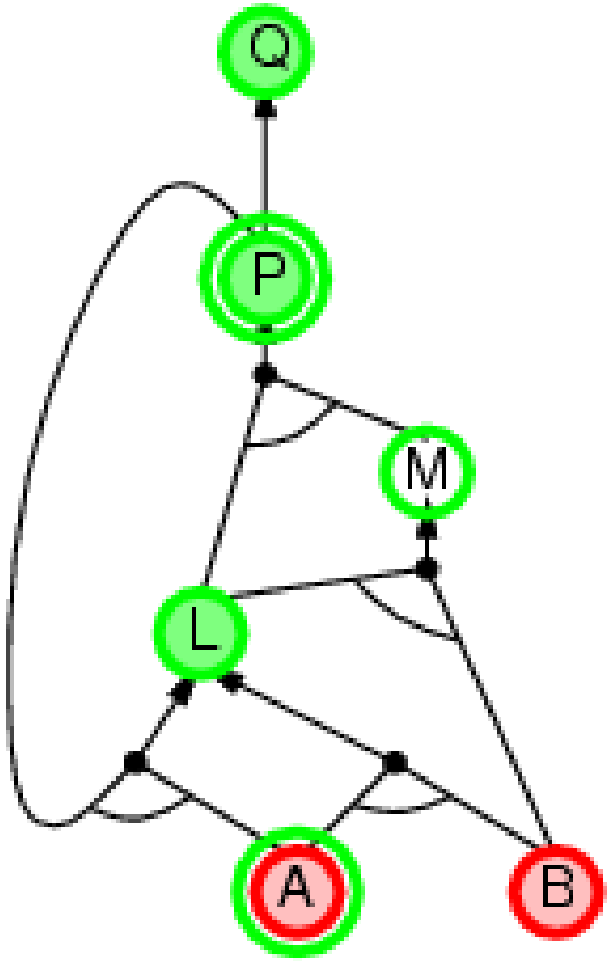
# Backward chaining example



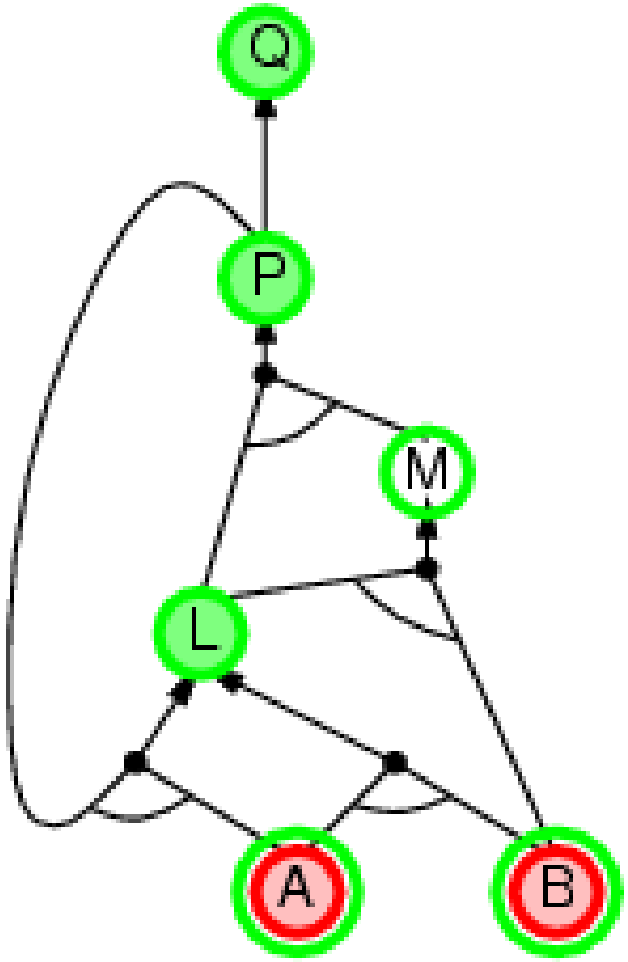
# Backward chaining example



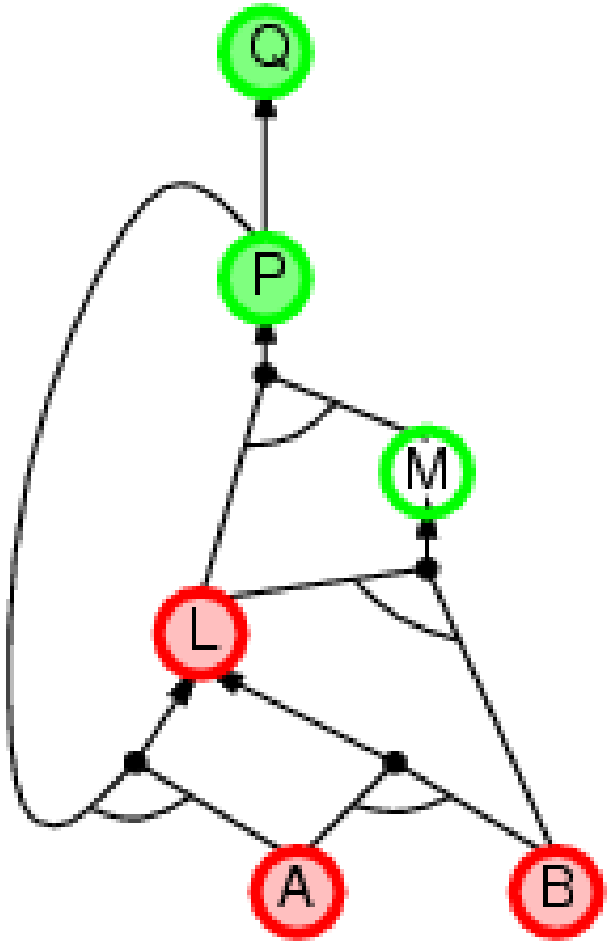
# Backward chaining example



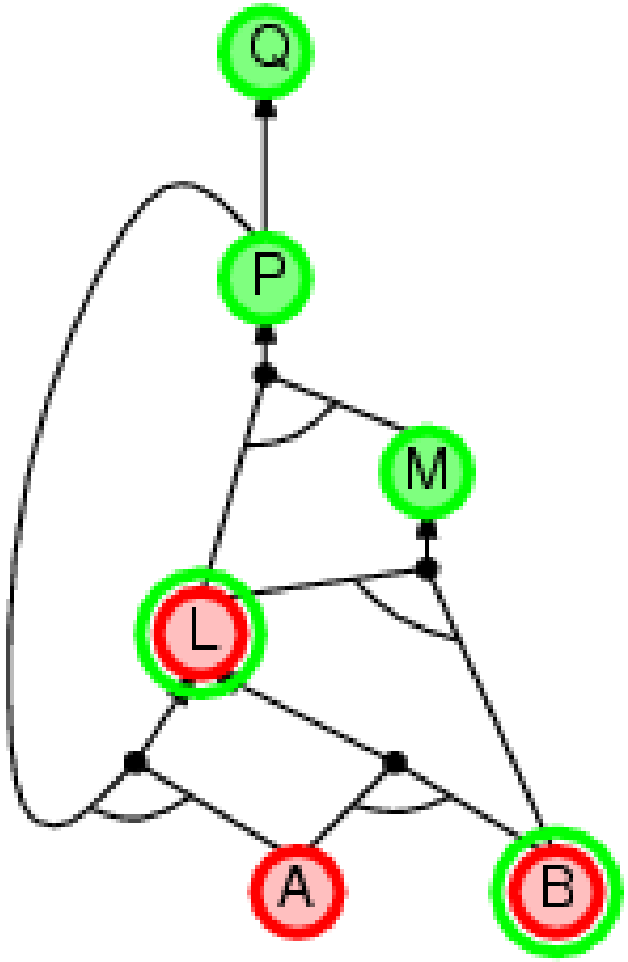
# Backward chaining example



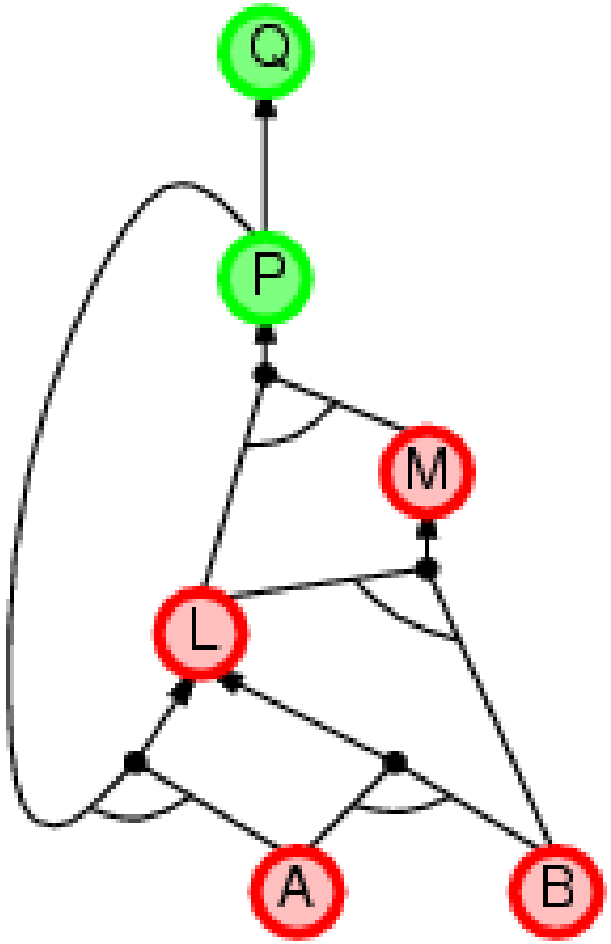
# Backward chaining example



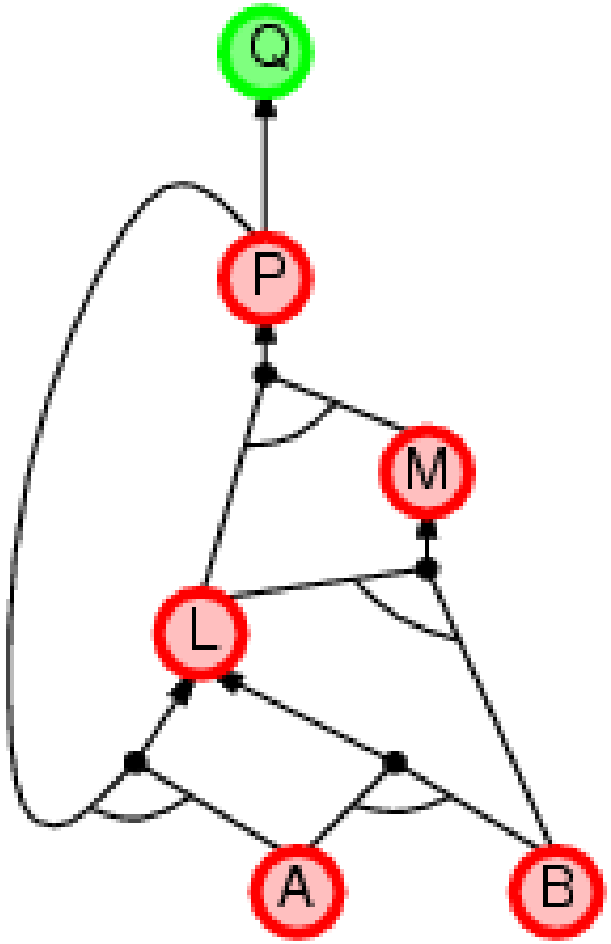
# Backward chaining example



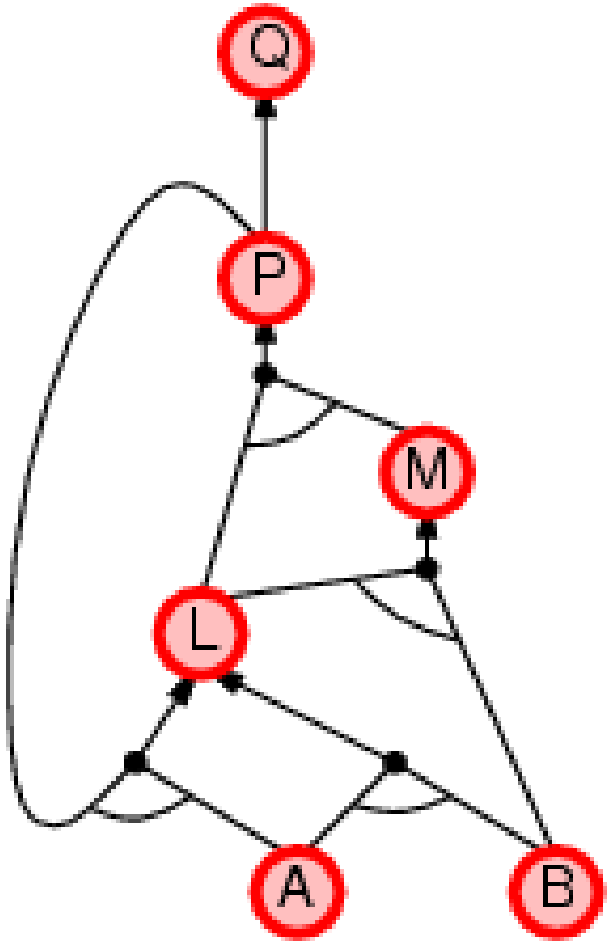
# Backward chaining example



# 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 model checking

Two families of efficient algorithms for propositional model checking:

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*])

# DPLL example

**(ØD** Ú **ØB** Ú **C)** Ù

**(B** Ú **ØA** Ú **ØC)** Ù

**(ØC** Ú **ØB** Ú **E)** Ù

**(E** Ú **ØD** Ú **B)** Ù

**(B** Ú **E** Ú **ØC)**

# DPLL example

**(ØA**Ú**ØB**Ú**ØC)** ù
**(ØA**Ú**B)** ù
**(ØA**Ú**C)** ù
**(ØB**Ú**C)** ù
**(ØB**Ú**A)** ù
**(ØC**Ú**A)** ù
**(ØC**Ú**B)**

# 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*

# WalkSAT example

**(ØD** Ú **ØB** Ú **C)** Ù

**(B** Ú **ØA** Ú **ØC)** Ù

**(ØC** Ú **ØB** Ú **E)** Ù

**(E** Ú **ØD** Ú **B)** Ù

**(B** Ú **E** Ú **ØC)**

# WalkSAT example

**(ØA**ÚØ**B**ÚØ**C)** Ù
**(ØA**Ú**B)** Ù
**(ØA**Ú**C)** Ù
**(ØB**Ú**C)** Ù
**(ØB**Ú**A)** Ù
**(ØC**Ú**A)** Ù
Ø**C**Ú**B**

# Hard satisfiability problems

- Consider random 3-CNF sentences. e.g.,

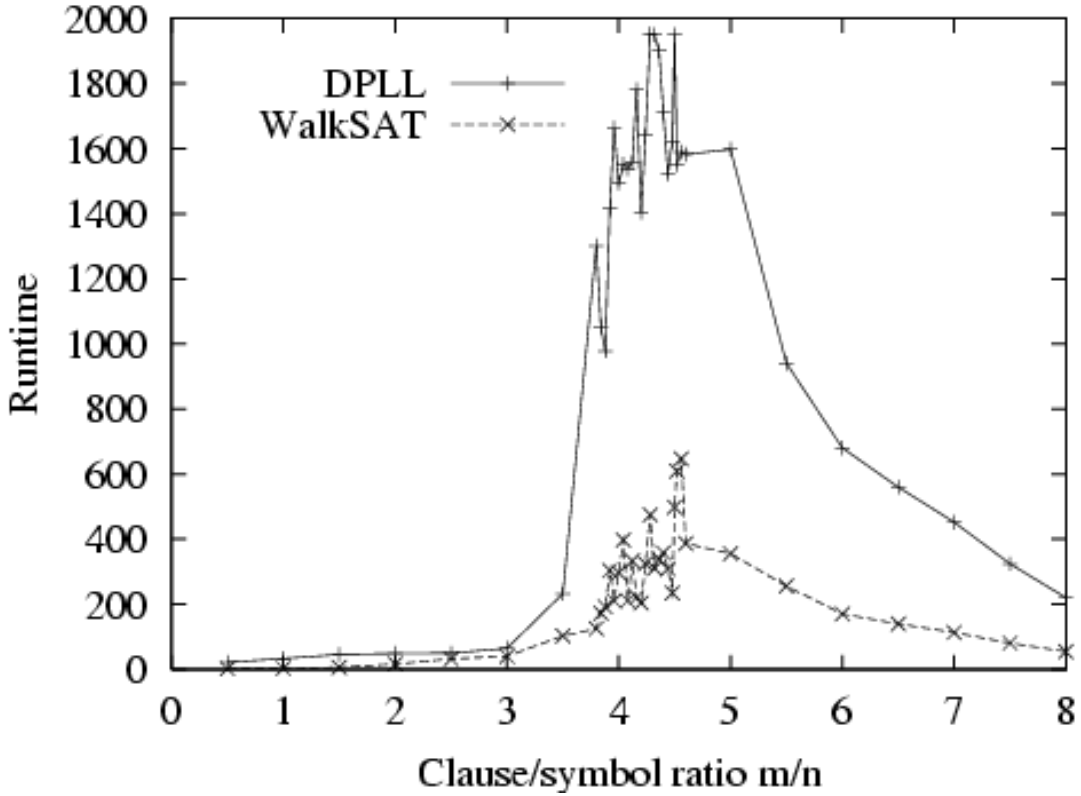
  $(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$

  $m$ = number of clauses

  $n$ = number of symbols

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

# Hard satisfiability problems




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

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

# Exercises

1. Prove

- $P \supset (Q \supset P)$
- $(P \supset (Q \supset R)) \supset ((P \supset Q) \supset (P \supset R))$
- $(Q \supset \neg P) \supset ((Q \supset P) \supset \neg Q)$

# Exercises

2. Convert into CNF

$$\neg[\ (\ (P \vee \neg Q) \supset R) \supset (P \wedge R)\ ]$$

# Exercises

3. Prove that “I win” having:

Heads, I win; tails, you lose.