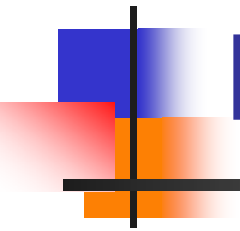
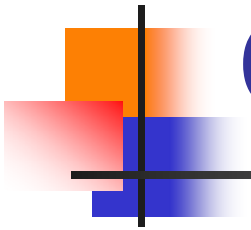


# Logical Agents

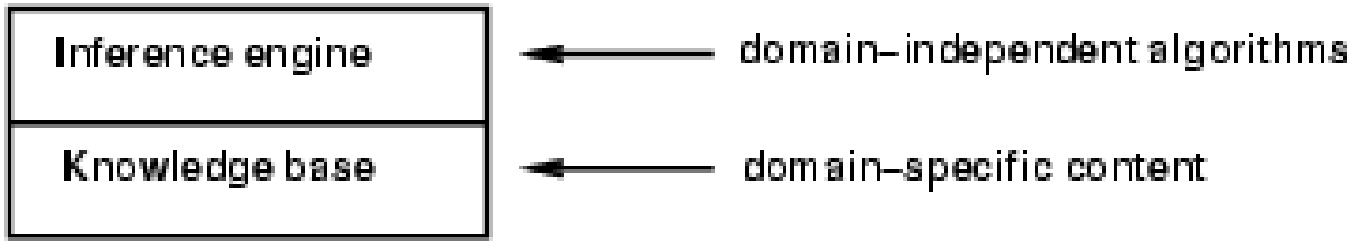
Chapter 7



# 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

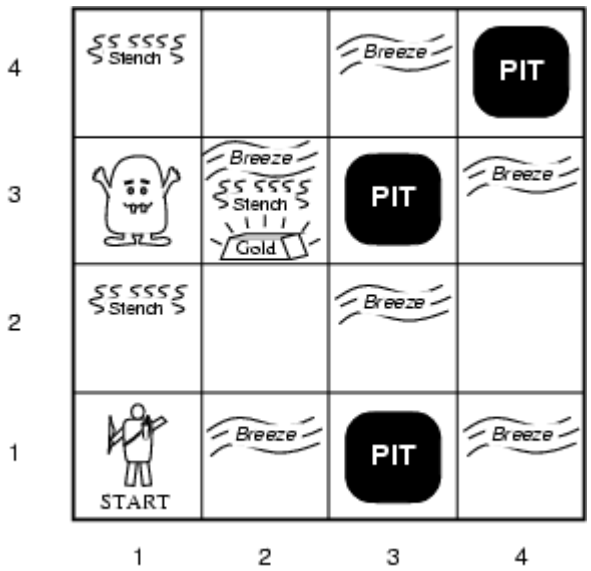
# A simple knowledge-based agent

```
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
```

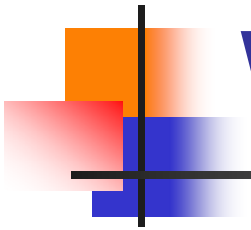
- The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

# 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



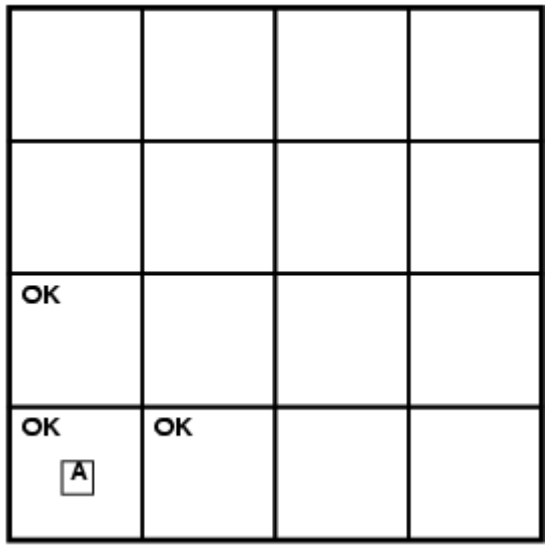

- Sensors: Stench, Breeze, Glitter, Bump, Scream
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot



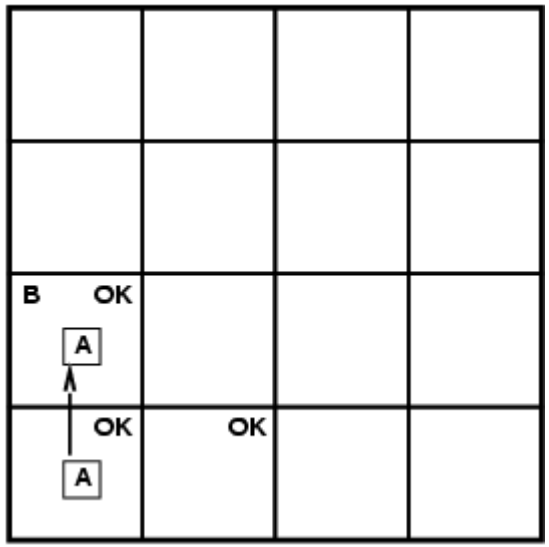
# Wumpus world characterization

- 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

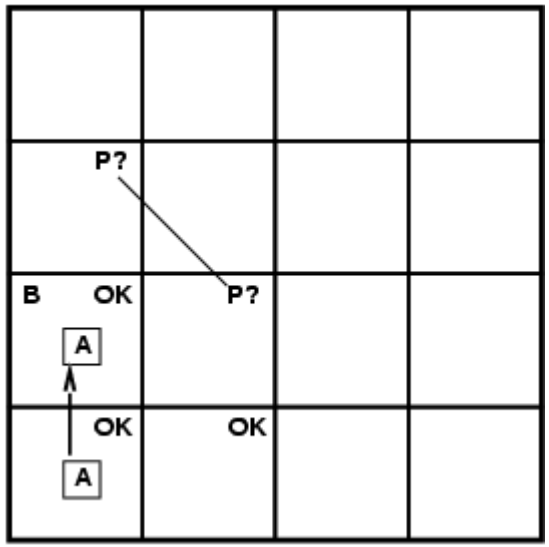
# Exploring a wumpus world



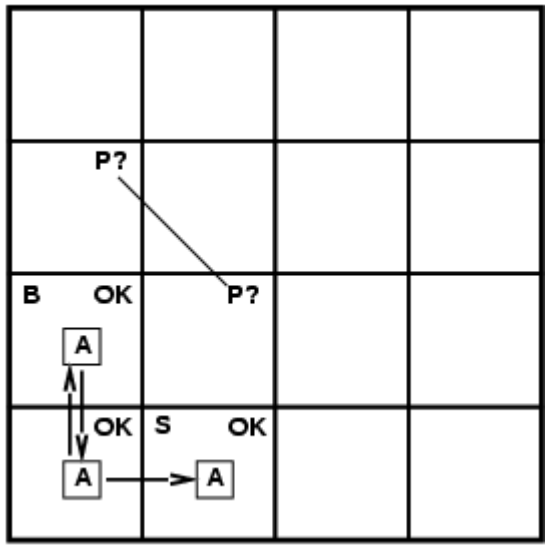
# Exploring a wumpus world



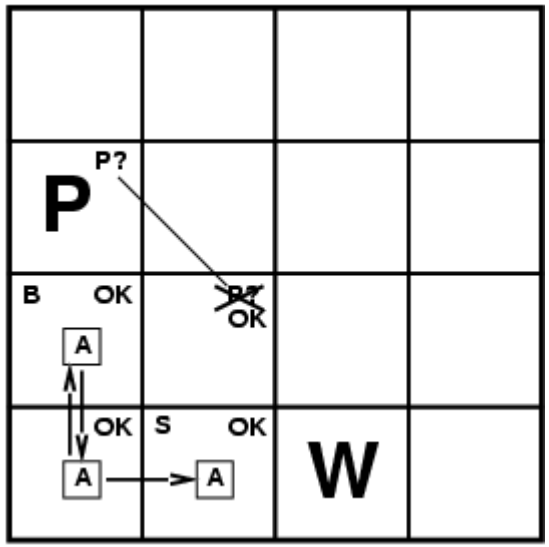
# Exploring a wumpus world



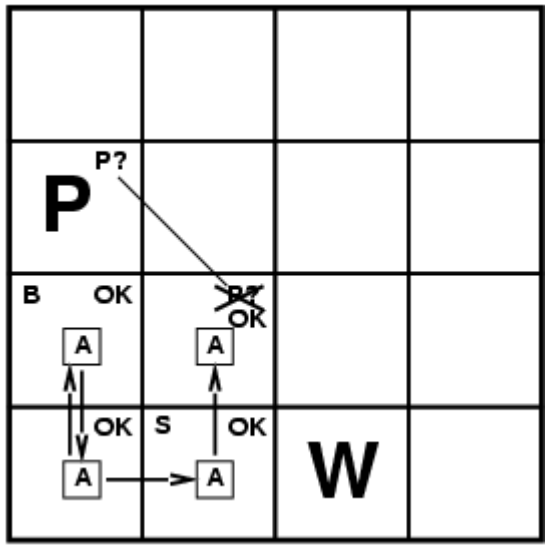
# Exploring a wumpus world



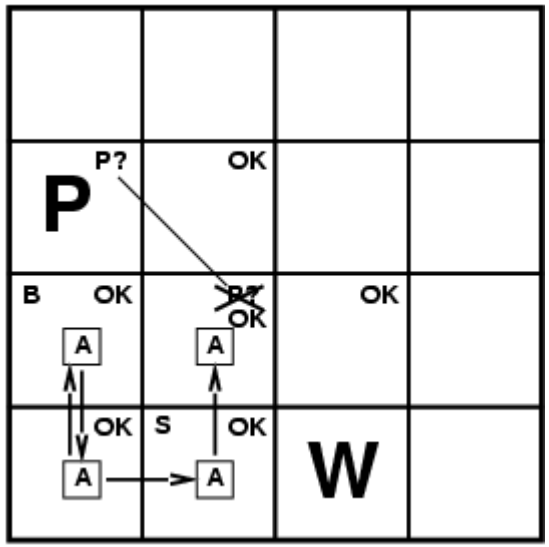
# Exploring a wumpus world



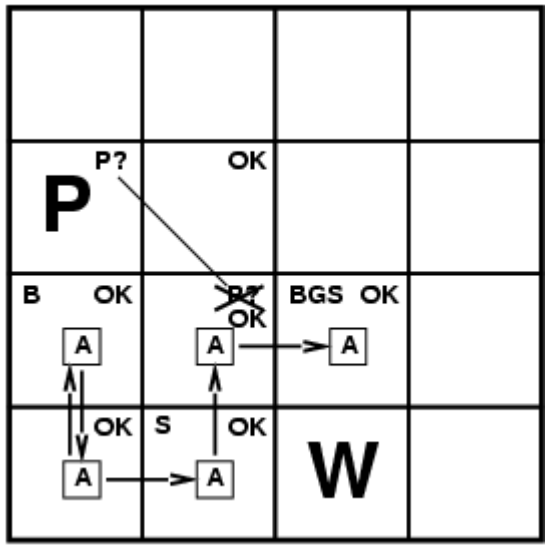
# Exploring a wumpus world



# 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
  - $x+2 \geq y$ 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

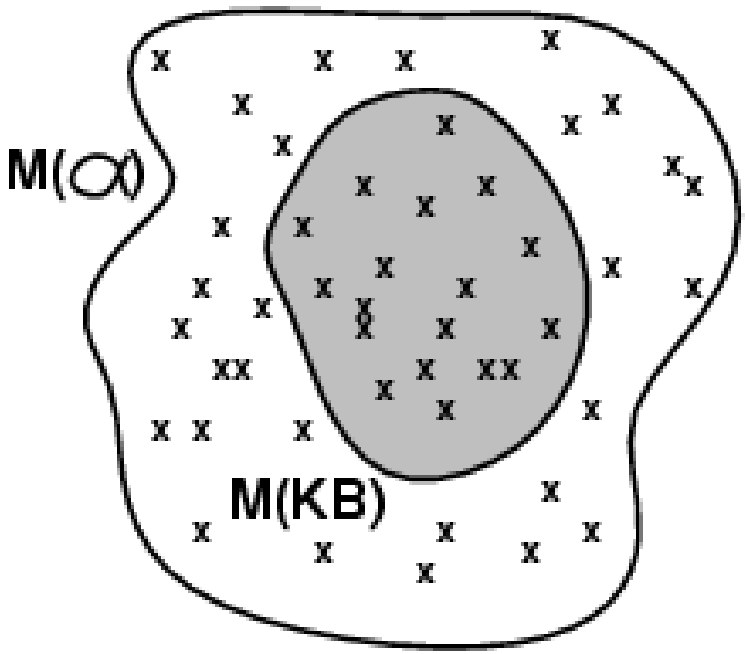
- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- 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

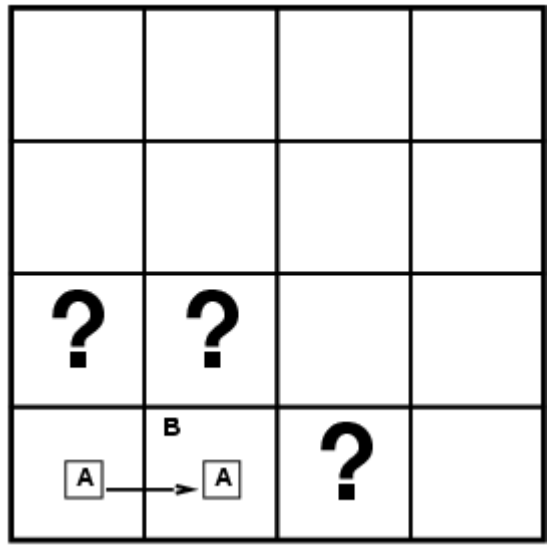
# Models

- 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

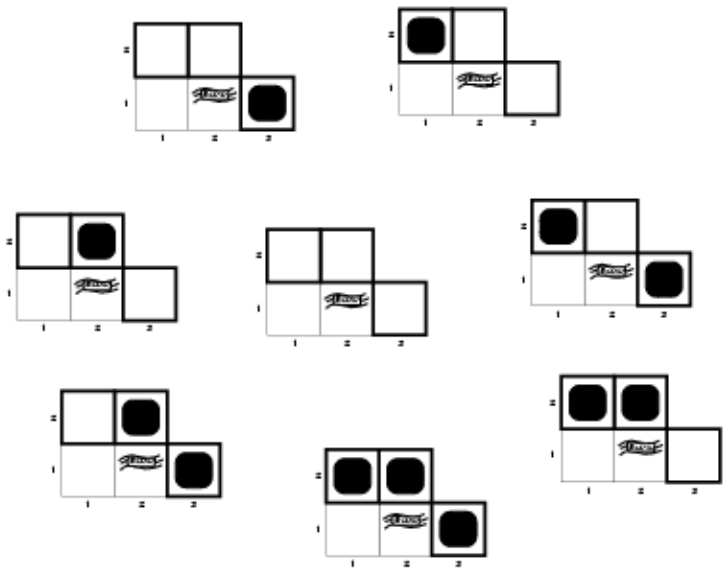
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]



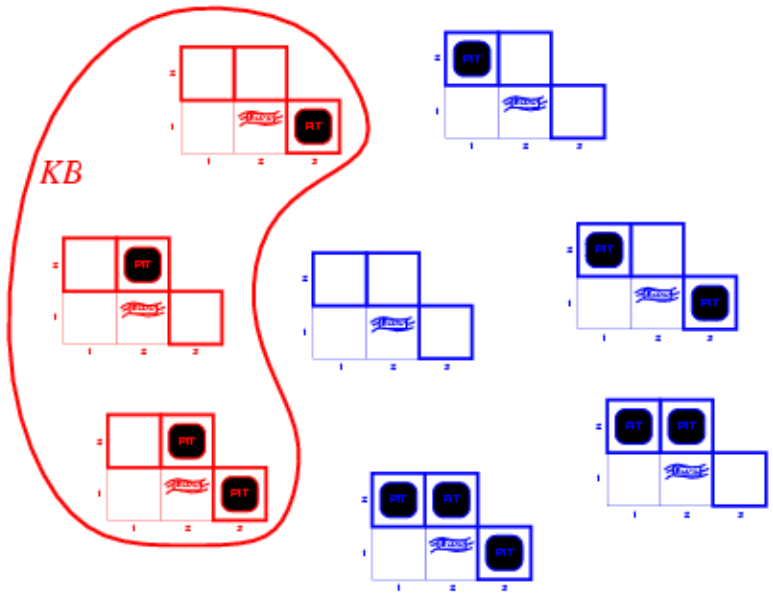

Consider possible models for *KB* assuming only pits

3 Boolean choices $\Rightarrow$ 8 possible models

# Wumpus models

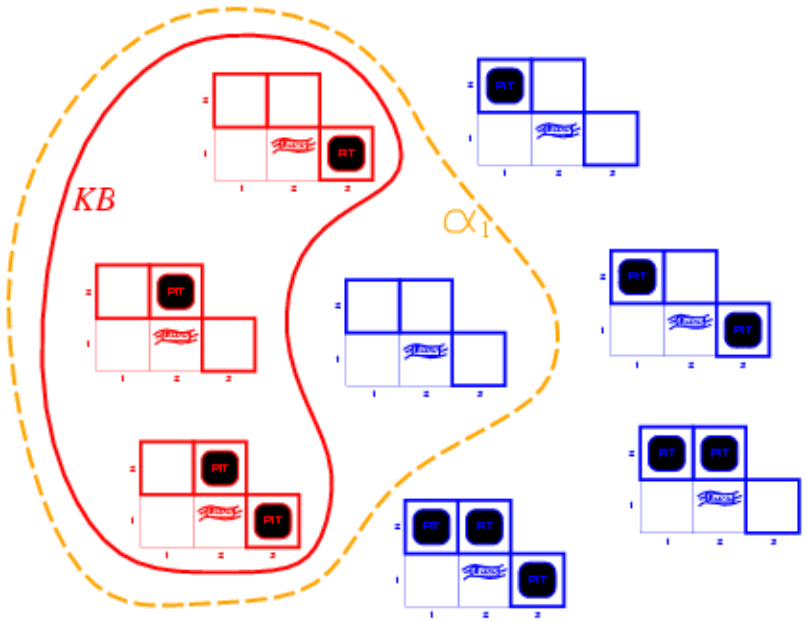


# Wumpus models



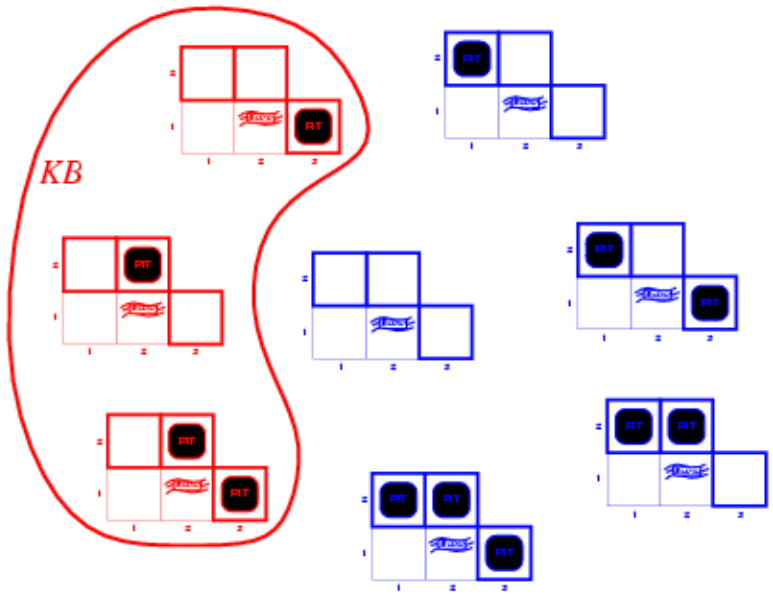

- *KB* = wumpus-world rules + observations

# Wumpus models



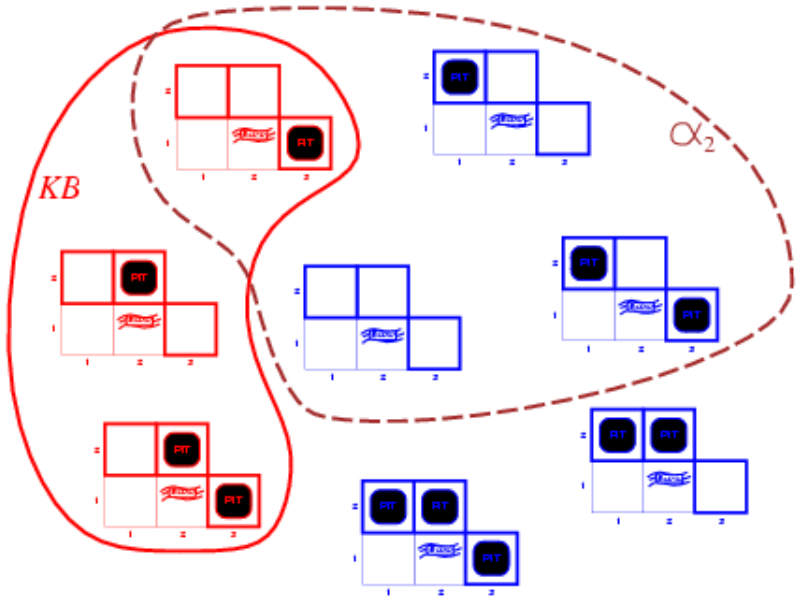

- *KB* = wumpus-world rules + observations
- $\alpha_1$ = "[1,2] is safe", *KB* $\models \alpha_1$, proved by model checking

# Wumpus models




- *KB* = wumpus-world rules + observations

# Wumpus models




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

# Inference

- Define: $KB \vdash_i \alpha$ = sentence $\alpha$ 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 $KB \vdash_i \alpha$
- 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 $P_1$, $P_2$ 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: 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

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

# 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"

# Truth tables for inference

| $B_{1,1}$ | $B_{2,1}$ | $P_{1,1}$ | $P_{1,2}$ | $P_{2,1}$ | $P_{2,2}$ | $P_{3,1}$ | $KB$ | $\alpha_1$ |
|---|---|---|---|---|---|---|---|---|
| $false$ | $false$ | $false$ | $false$ | $false$ | $false$ | $false$ | $false$ | $true$ |
| $false$ | $false$ | $false$ | $false$ | $false$ | $false$ | $true$ | $false$ | $true$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $false$ | $true$ | $false$ | $false$ | $false$ | $false$ | $false$ | $false$ | $true$ |
| $false$ | $true$ | $false$ | $false$ | $false$ | $false$ | $true$ | $\underline{true}$ | $\underline{true}$ |
| $false$ | $true$ | $false$ | $false$ | $false$ | $true$ | $false$ | $\underline{true}$ | $\underline{true}$ |
| $false$ | $true$ | $false$ | $false$ | $false$ | $true$ | $true$ | $\underline{true}$ | $\underline{true}$ |
| $false$ | $true$ | $false$ | $false$ | $true$ | $false$ | $false$ | $false$ | $true$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $true$ | $true$ | $true$ | $true$ | $true$ | $true$ | $true$ | $false$ | $false$ |

# Inference by enumeration

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

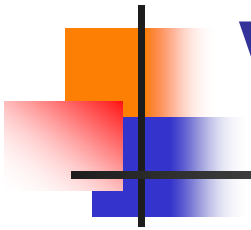
```
function TT-ENTAILS?(KB, α) returns true or false

    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: α ≡ ß iff α ╞ β and β ╞ α

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



# 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

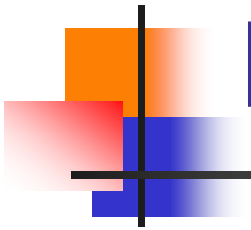
e.g., $A \vee B$, $C$

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



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

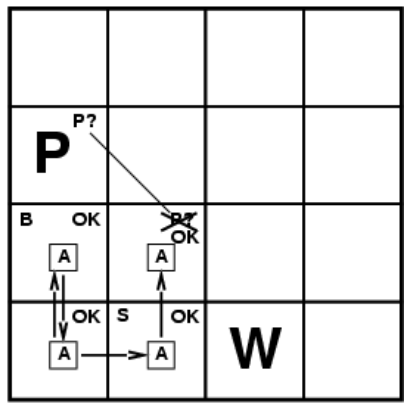
- Resolution inference rule (for CNF):

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

where $\ell_i$ 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 and complete for propositional logic



# Resolution

Soundness of resolution inference rule:

$$\frac{\neg(\ell_i \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k) \Rightarrow \ell_i \qquad \neg m_j \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}{\neg(\ell_i \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k) \Rightarrow (m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

where $\ell_i$ and $m_j$ are complementary literals.

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

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

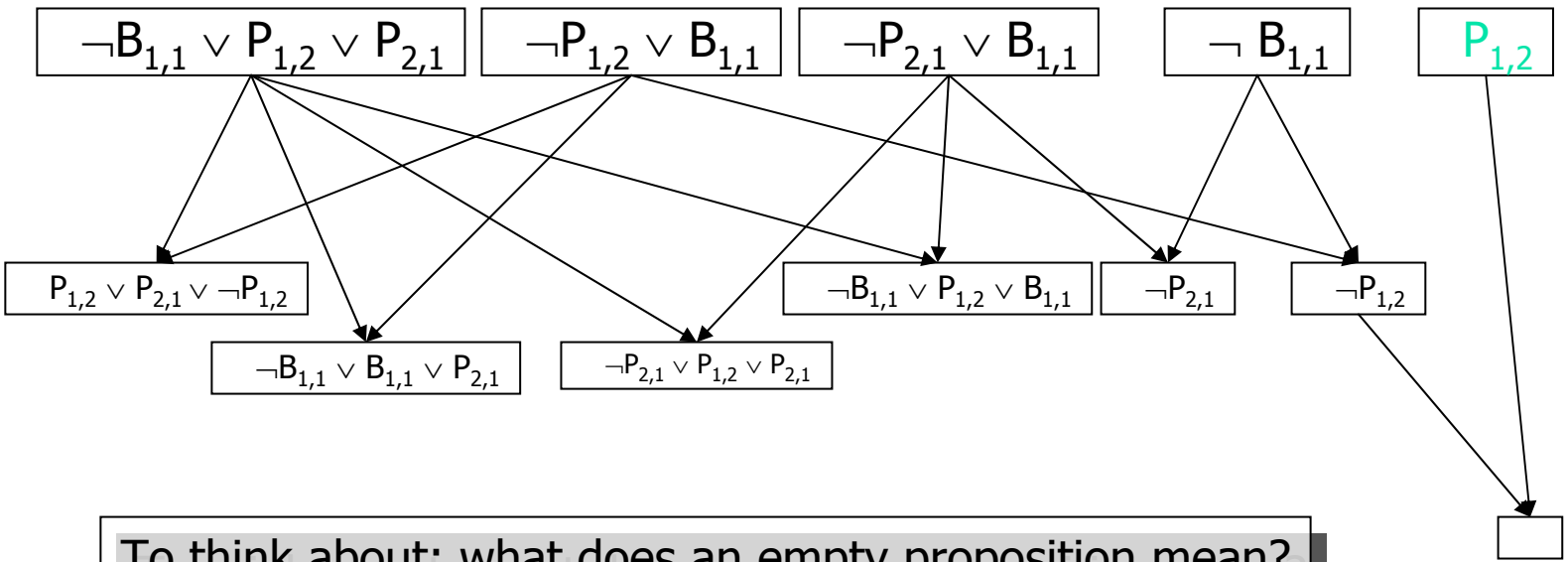
# Resolution algorithm

- 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 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

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$ (negate the premise for proof by refutation)




To think about: what does an empty proposition mean?

# 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

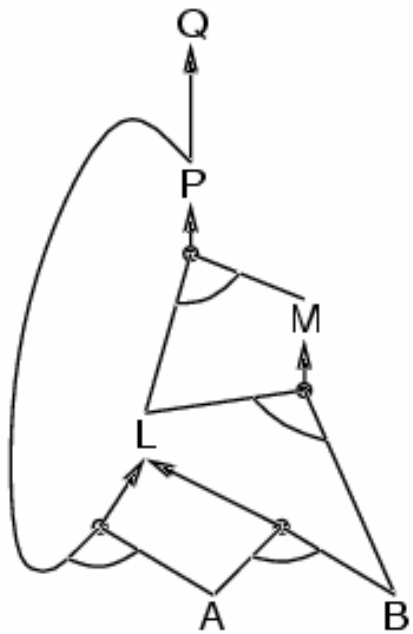
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- 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

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

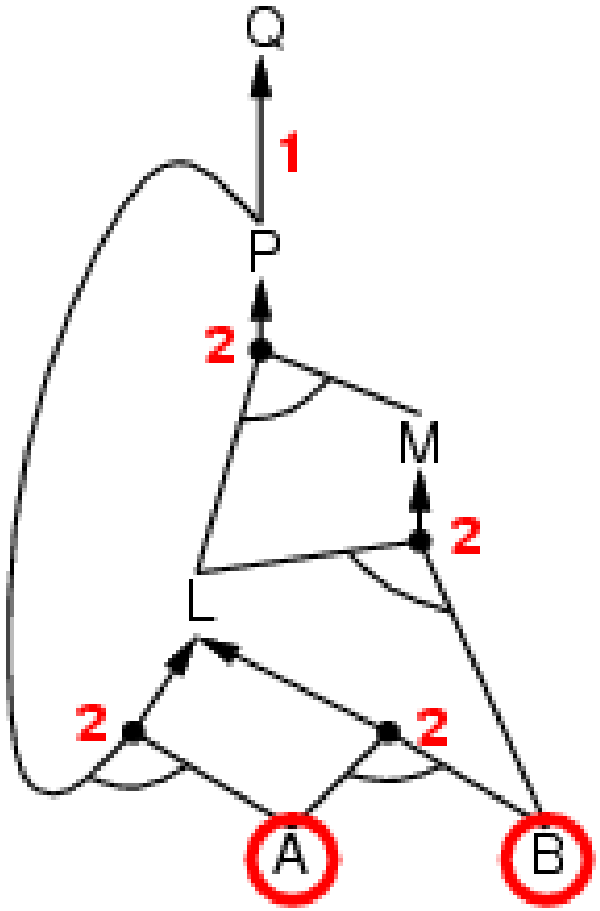


# Forward chaining algorithm

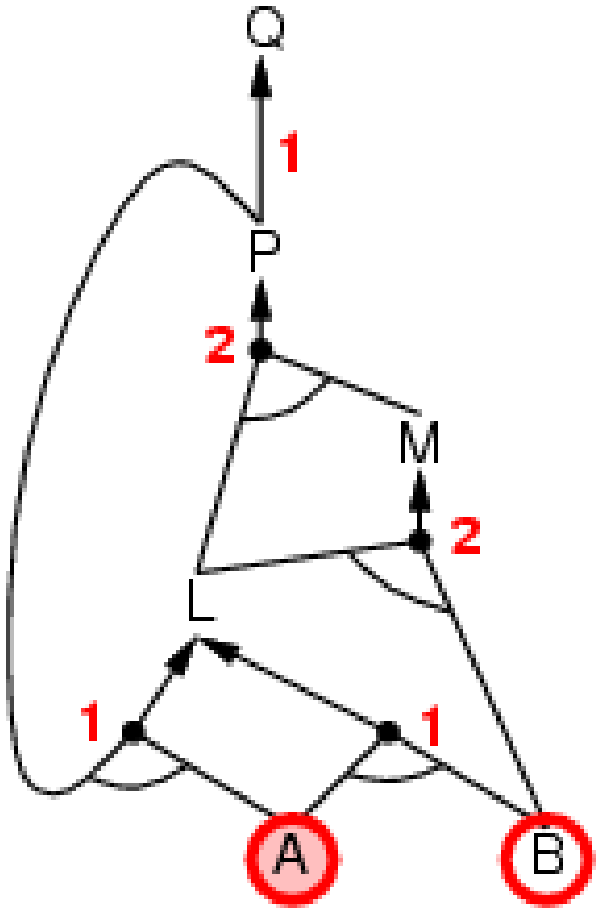
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

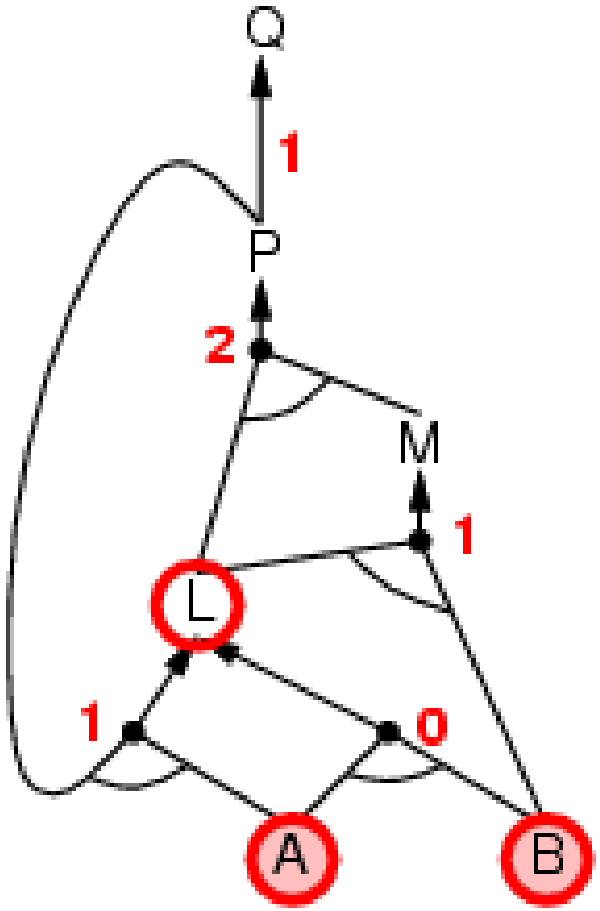
# Forward chaining example



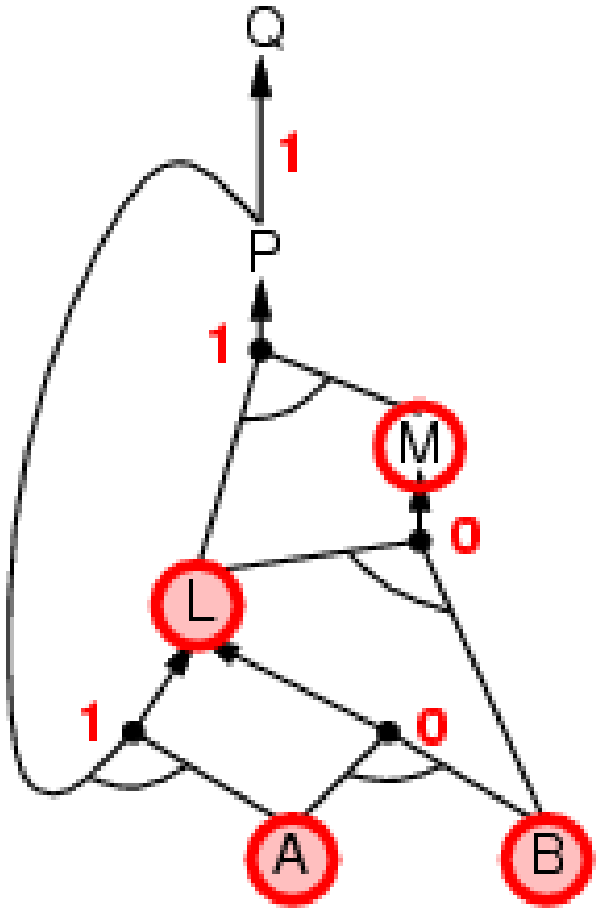
# Forward chaining example



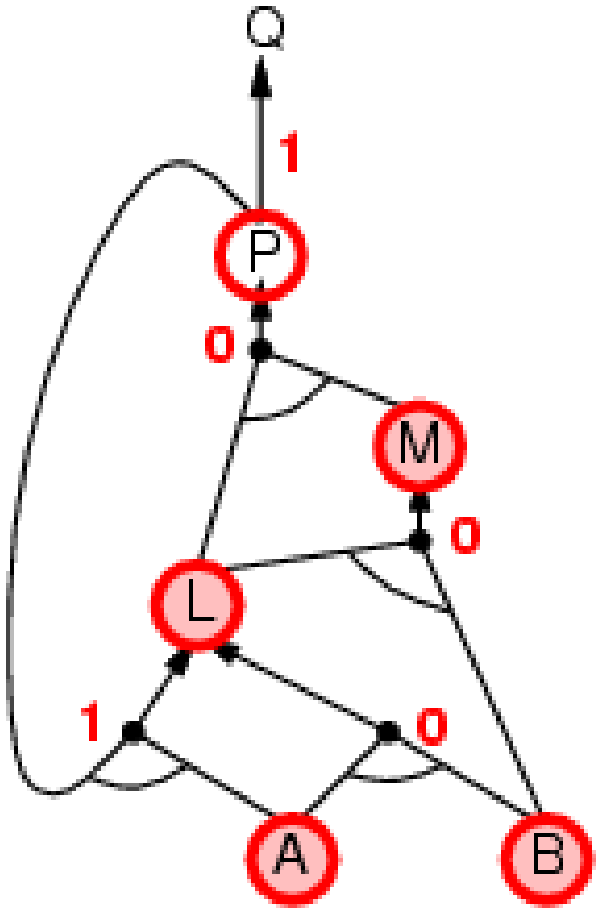
# Forward chaining example



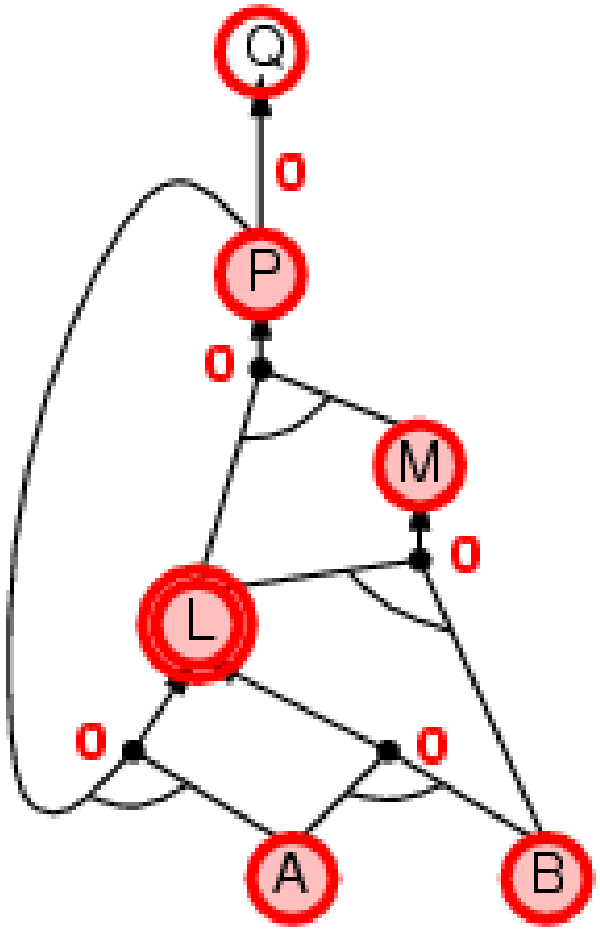
# Forward chaining example



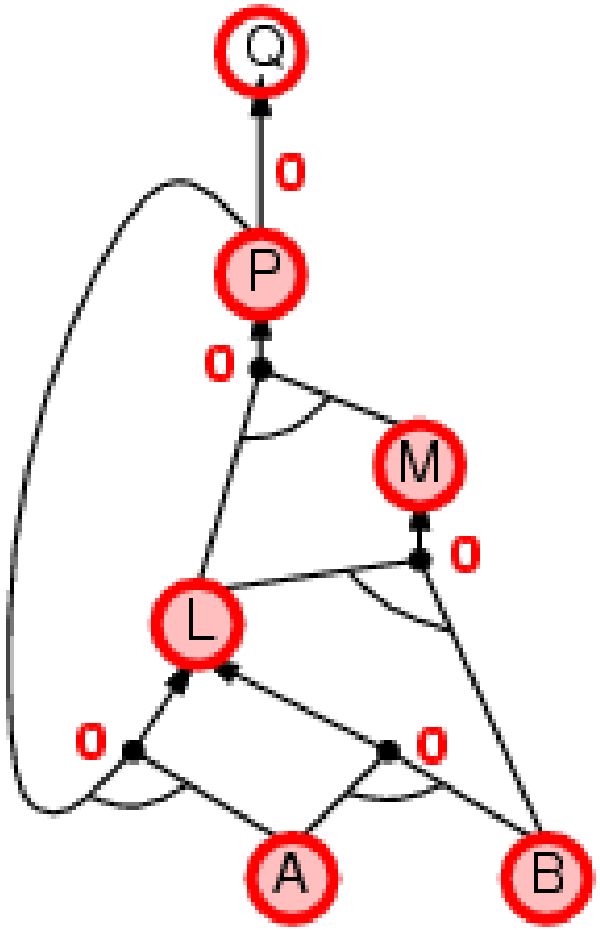
# Forward chaining example



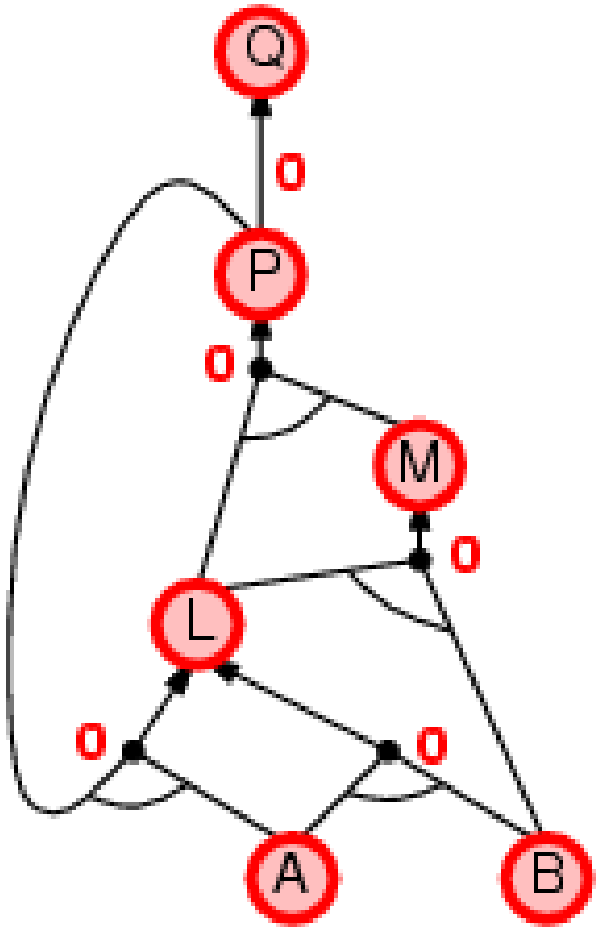
# Forward chaining example



# Forward chaining example



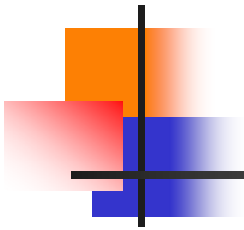
# Forward chaining example



# 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

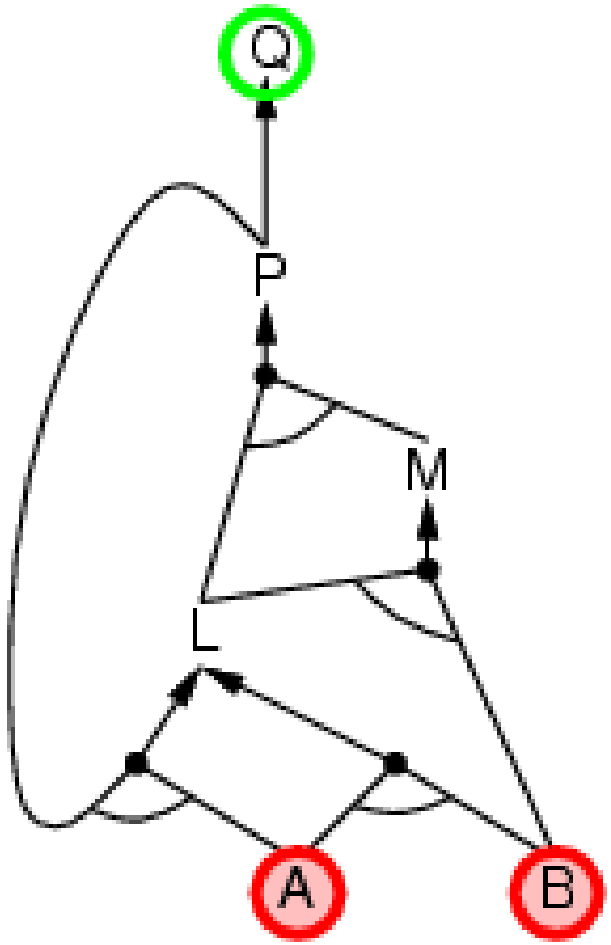
prove by BC all premises of some rule concluding *q*

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

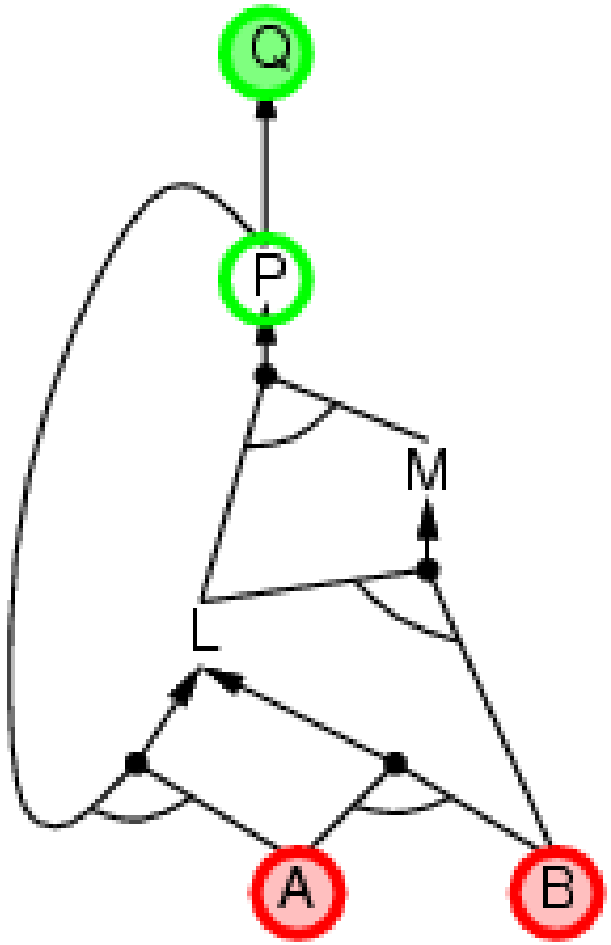
Avoid repeated work: check if new subgoal

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

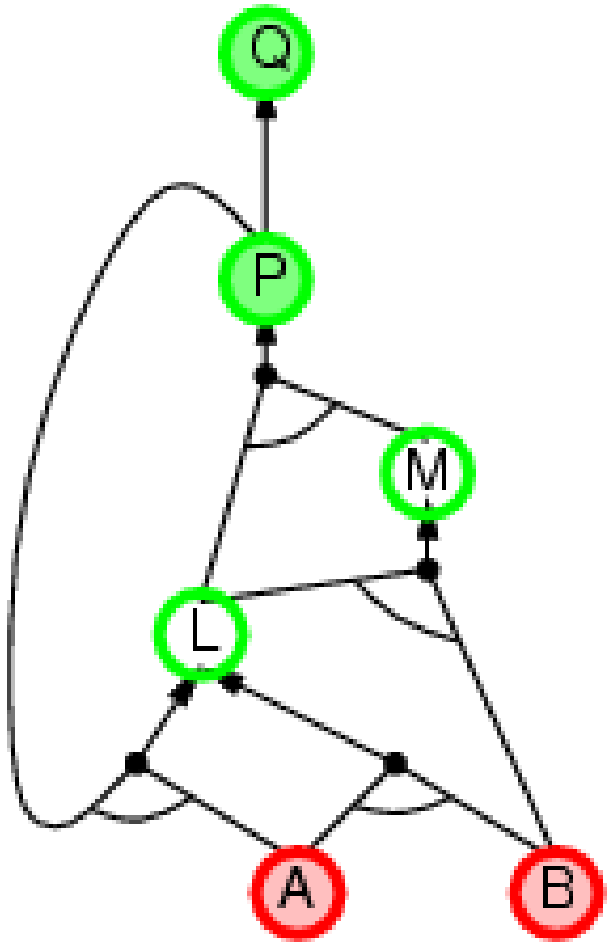
# Backward chaining example



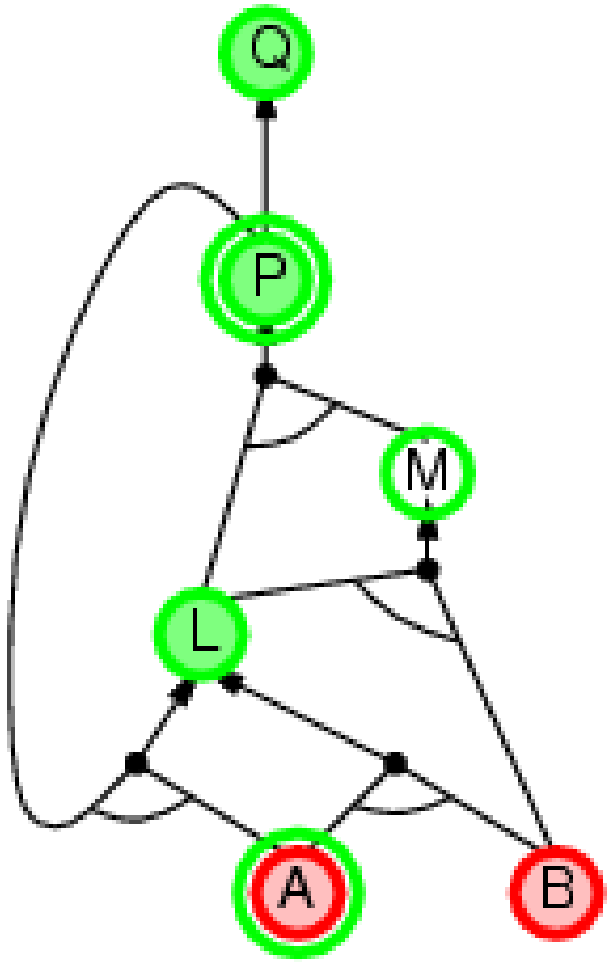
# Backward chaining example



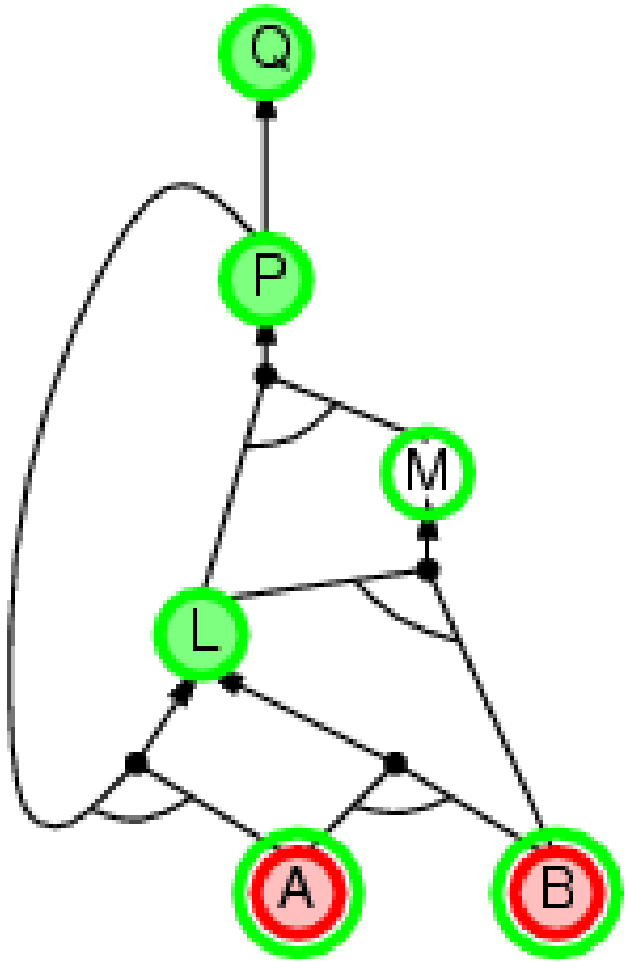
# Backward chaining example



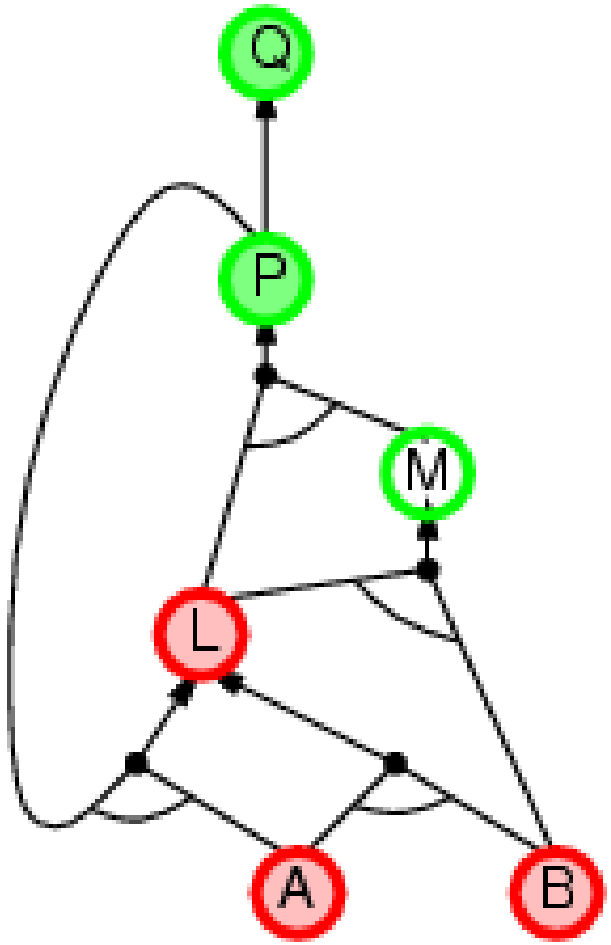
# Backward chaining example



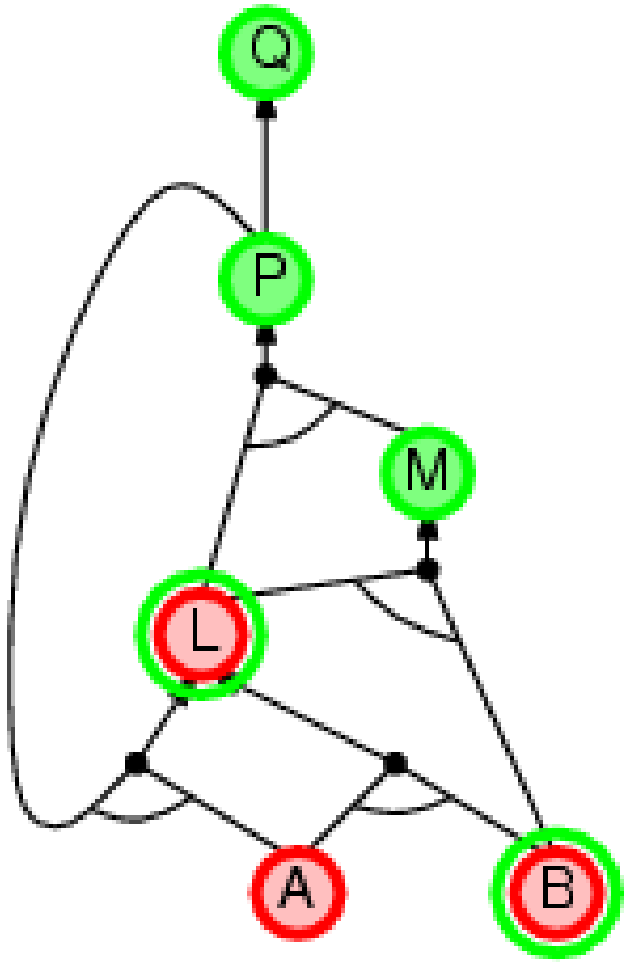
# Backward chaining example



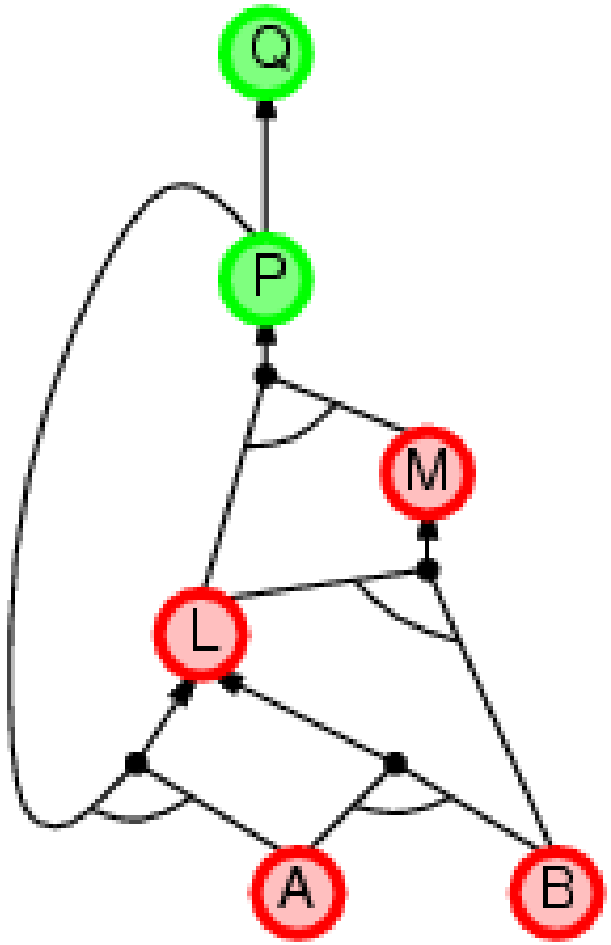
# Backward chaining example



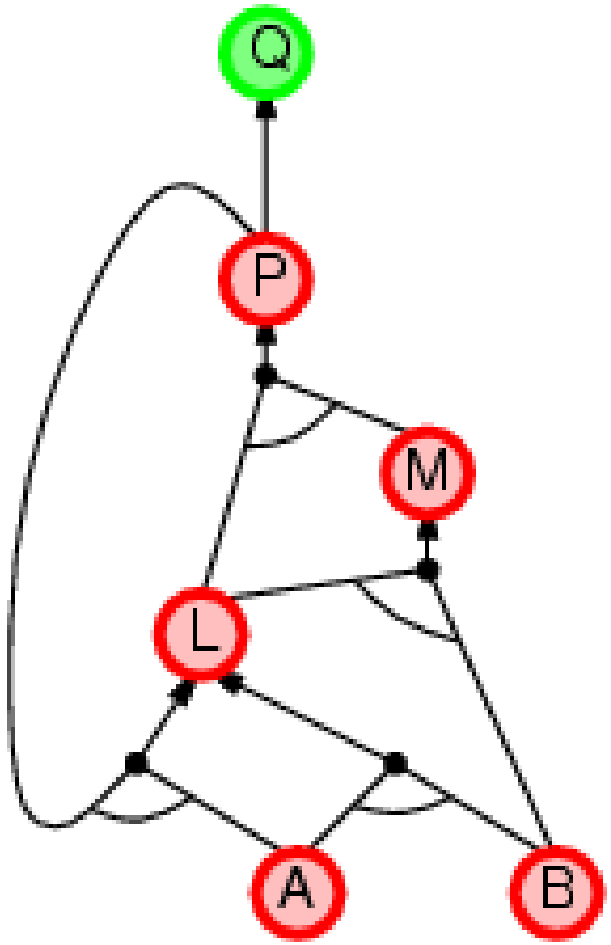
# Backward chaining example



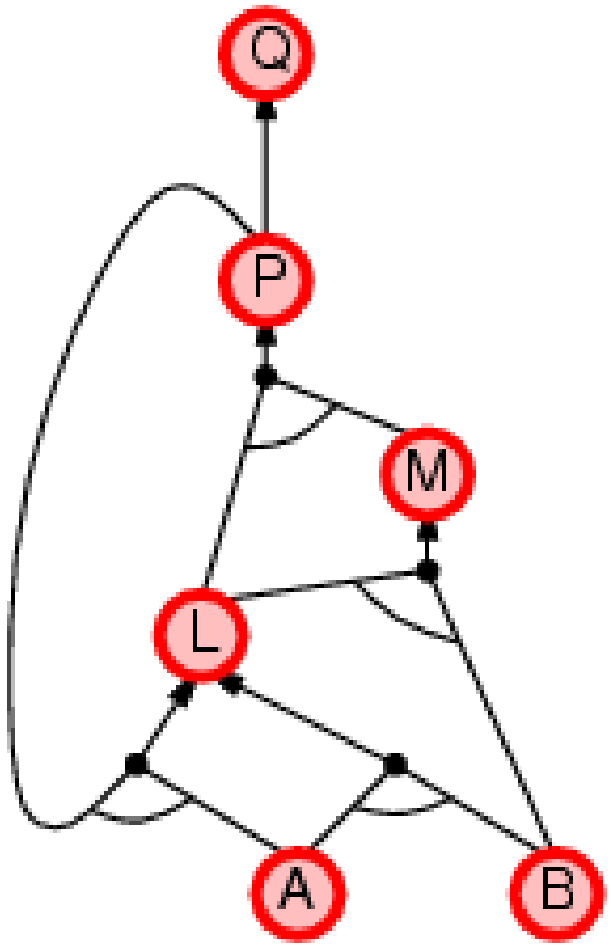
# Backward chaining example

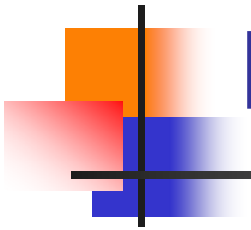


# Backward chaining example



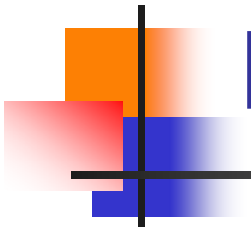
# Backward chaining example





# 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

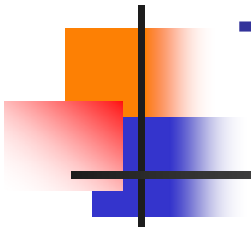


# Efficient propositional inference

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

$clauses \leftarrow$ the set of clauses in the CNF representation of $s$

$symbols \leftarrow$ 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*

$P$, $value \leftarrow$ FIND-PURE-SYMBOL($symbols$, $clauses$, $model$)

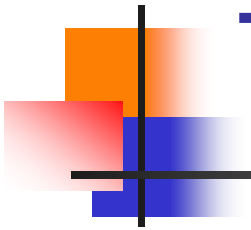
**if** $P$ is non-null **then return** DPLL($clauses$, $symbols$–$P$, $[P = value | model]$)

$P$, $value \leftarrow$ FIND-UNIT-CLAUSE($clauses$, $model$)

**if** $P$ is non-null **then return** DPLL($clauses$, $symbols$–$P$, $[P = value | model]$)

$P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ 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.,

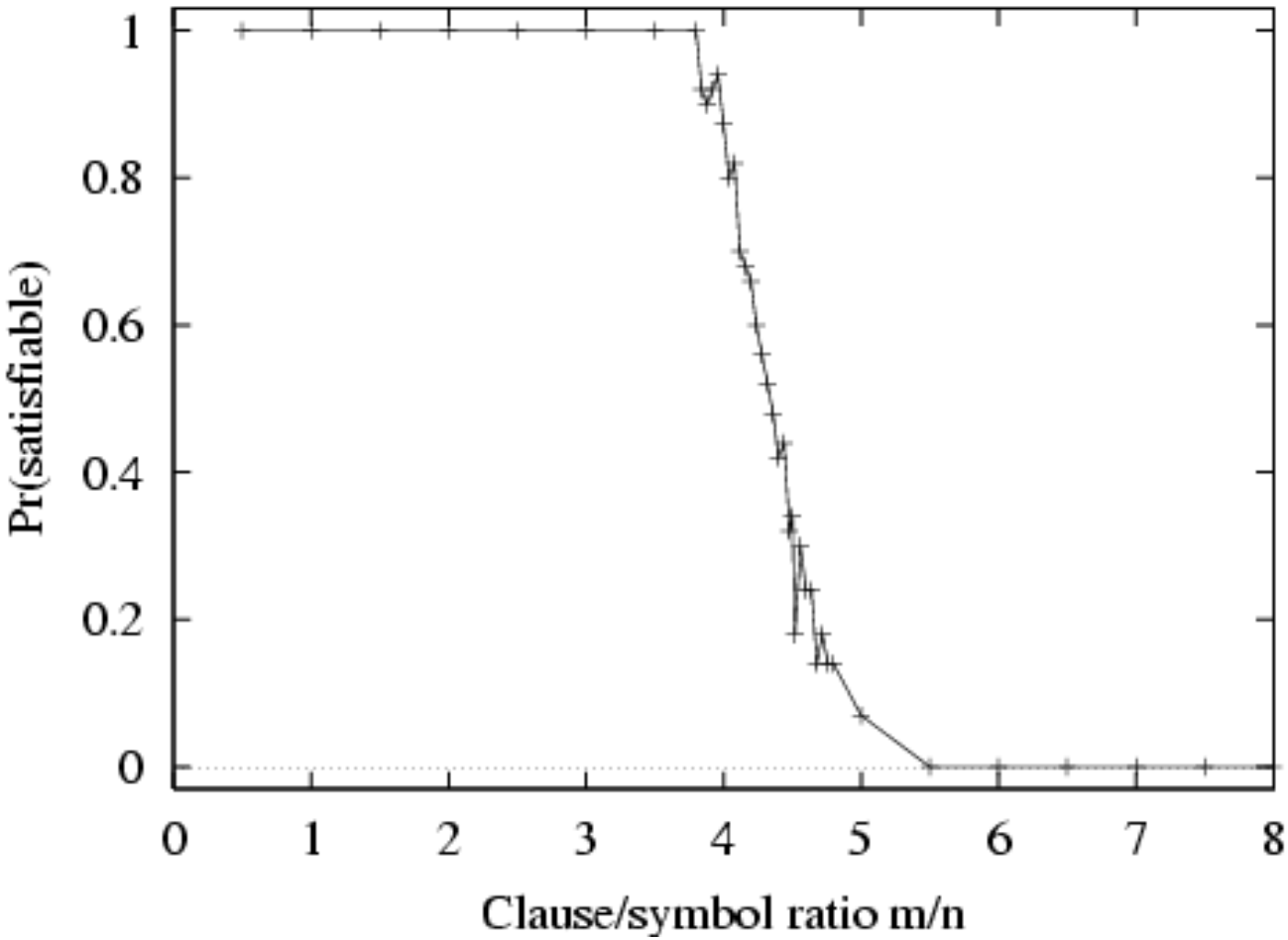
  $(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$

  $m$ = number of clauses

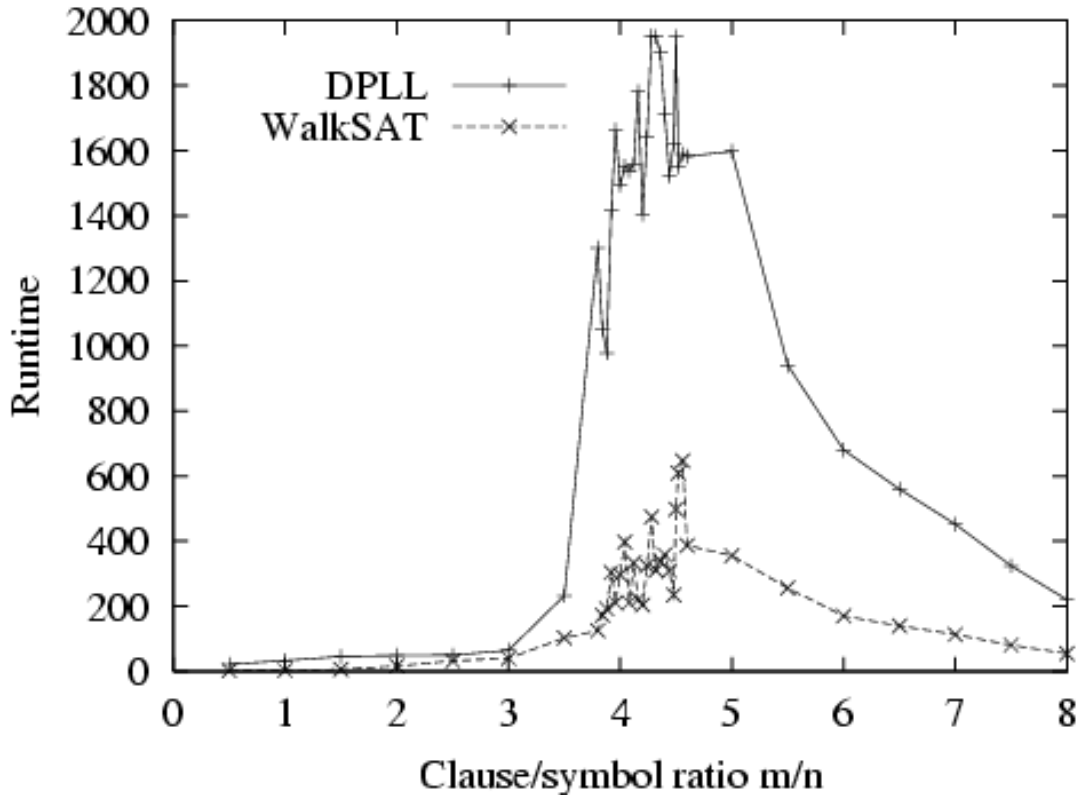
  $n$ = number of symbols

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

# Hard satisfiability problems



# 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

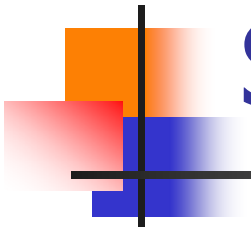
update *x*, *y*, *orientation*, *visited* based on *action*
**if** *stench* **then** TELL($KB, S_{x,y}$) **else** TELL($KB, \neg S_{x,y}$)
**if** *breeze* **then** TELL($KB, B_{x,y}$) **else** TELL($KB, \neg B_{x,y}$)
**if** *glitter* **then** *action* ← *grab*
**else if** *plan* is nonempty then *action* ← POP(*plan*)
**else if** for some fringe square [*i*,*j*], ASK($KB, (\neg P_{i,j} \wedge \neg W_{i,j})$) is *true* **or**
for some fringe square [*i*,*j*], ASK($KB, (P_{i,j} \vee W_{i,j})$) 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 $[x,y]$,

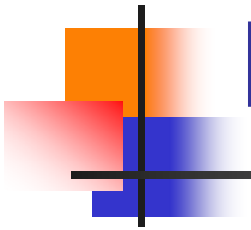
$$L^{t}_{x,y} \wedge \mathit{FacingRight}^{t} \wedge \mathit{Forward}^{t} \Rightarrow L^{t}_{x+1,y}$$

- 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



# Midterm test

- Five questions, first hour of class (be on time!)
- Topics to be covered (**CSP, prop logic** not on the midterm):
  - Chapter 2 – Agents
  - Chapter 3 – Uninformed Search
  - Chapter 4 – Informed Search
    - Not including the parts of 4.1 (memory-bounded heuristic search) and 4.5
  - Chapter 6 – Adversarial Search
    - Not including 6.5 (games with chance)