Logical Agents

Reading: Russell's Chapter 7

Environments

To design an agent we must specify its task environment.

PEAS description of the task environment:

- Performance
- Environment
- Actuators
- Sensors

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

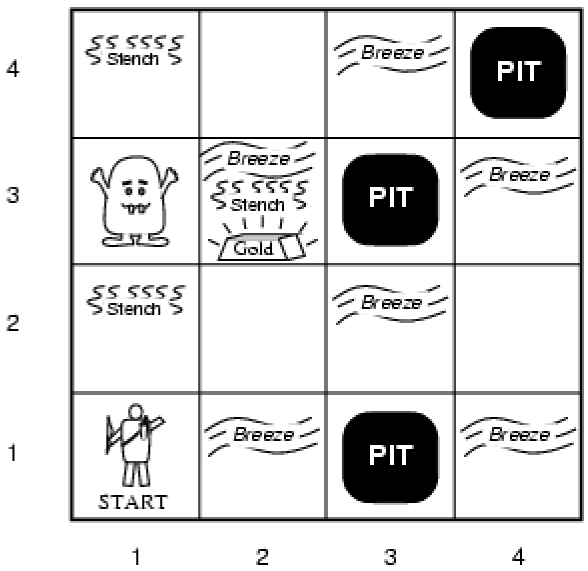
Glitter iff gold is in the same square

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

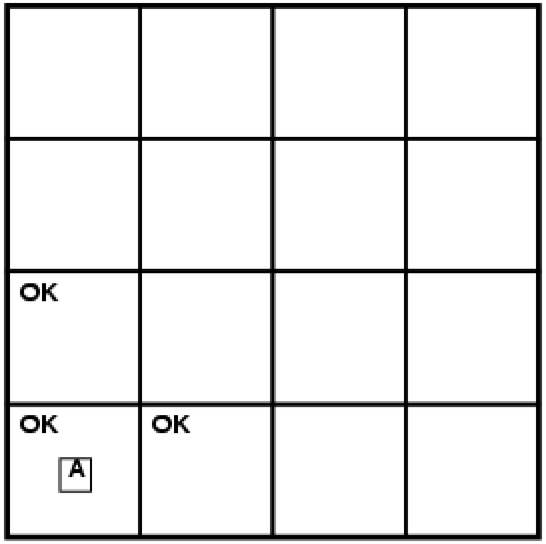
Static Yes – Wumpus and Pits do not move

Discrete Yes

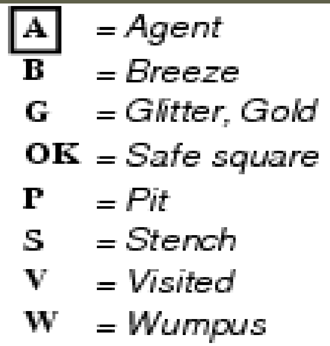
Single-agent? Yes – Wumpus is essentially a natural feature

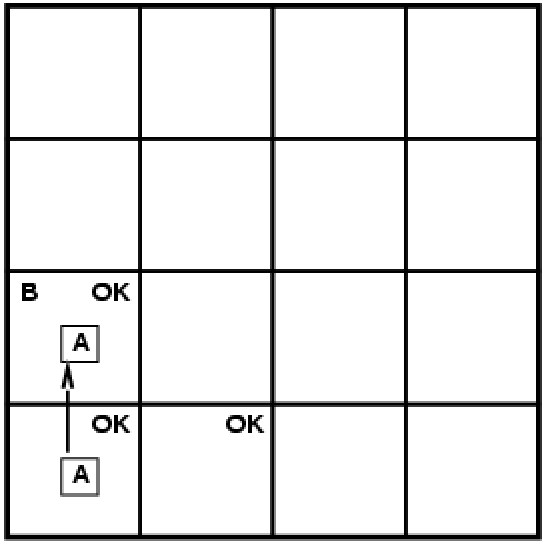
Exploring a wumpus world

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*



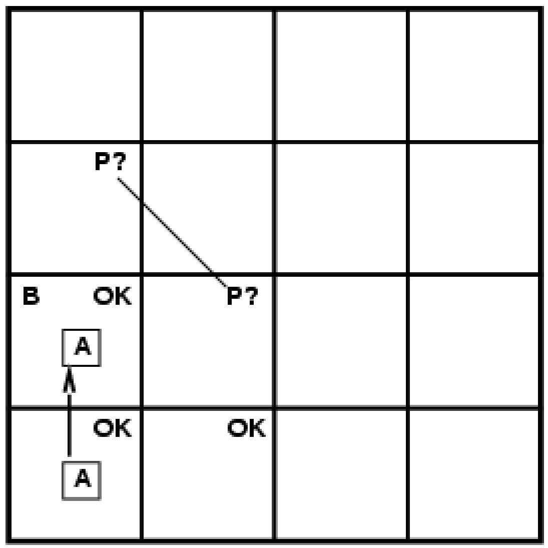
Exploring a wumpus world



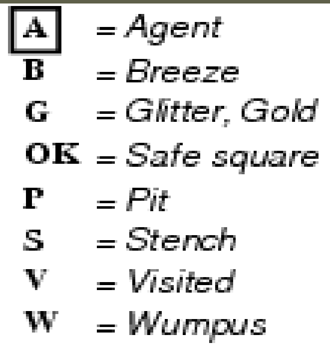


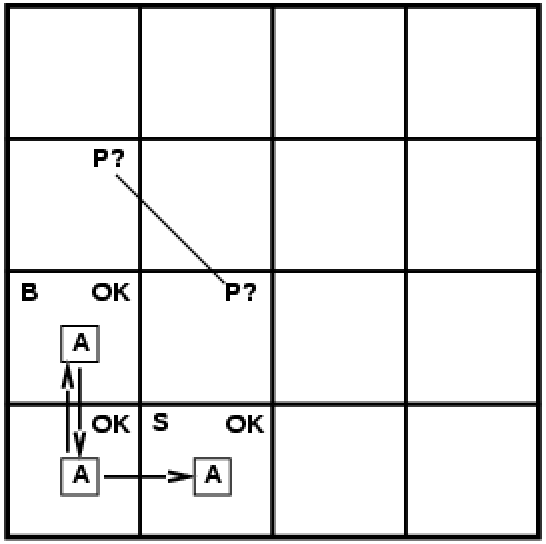
Exploring a wumpus world

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*



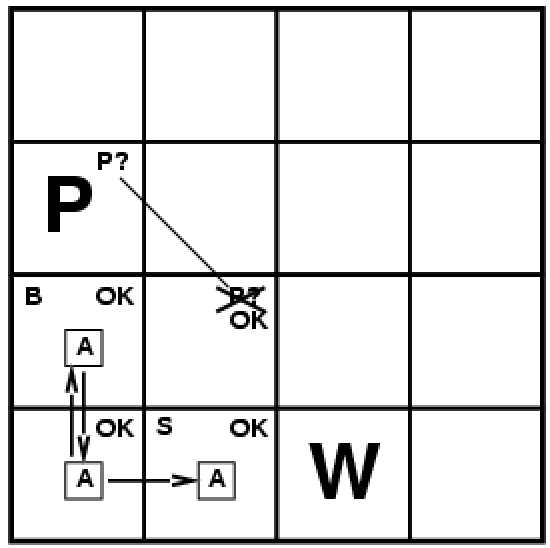
Exploring a wumpus world



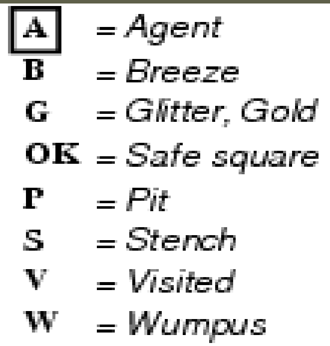


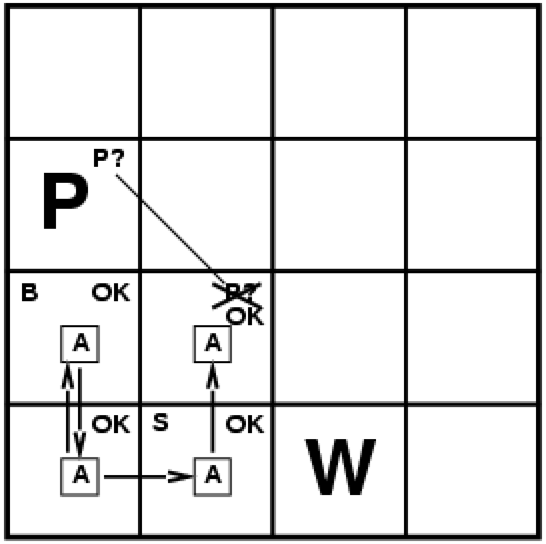
Exploring a wumpus world

$\boxed{A}$ = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*



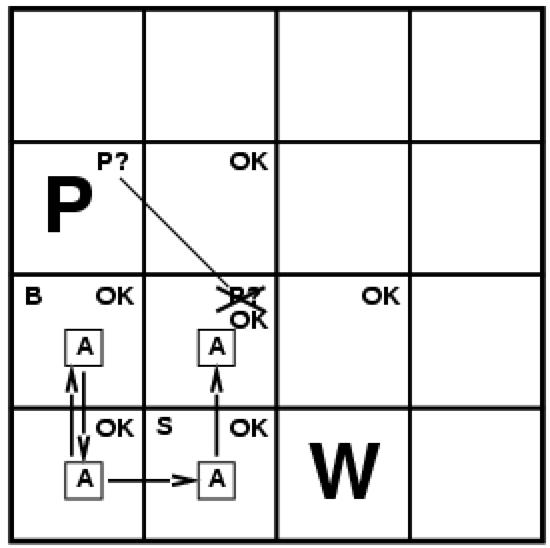
Exploring a wumpus world





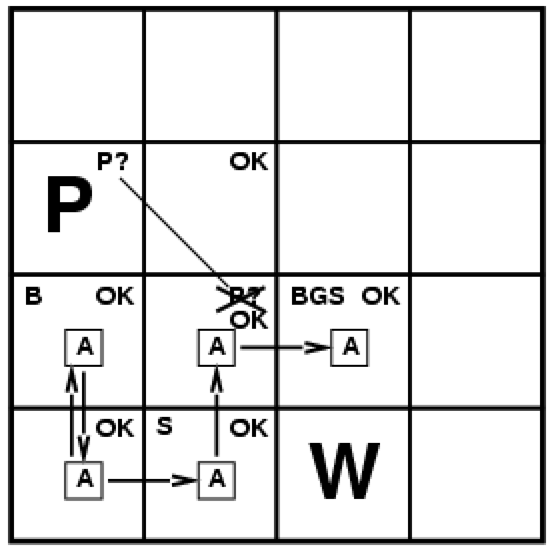
Exploring a wumpus world

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus



Exploring a wumpus world

- **A** = *Agent*
- **B** = *Breeze*
- **G** = *Glitter, Gold*
- **OK** = *Safe square*
- **P** = *Pit*
- **S** = *Stench*
- **V** = *Visited*
- **W** = *Wumpus*



Logic

- When we have too many states, we want a convenient way of dealing with sets of states.
- The sentence “It’s raining” stands for all the states of the world in which it is raining.
- Logic provides a way of manipulating big collections of sets by manipulating short descriptions instead.

Logic

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 $x = 0$, $y = 6$

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)

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

Semantics

Meaning of a sentence is truth value {**t, f**}

Interpretation is an assignment of truth values to the propositional variables

$\models_i \alpha$ [Sentence α is **t** in interpretation i]

$\not\models_i \alpha$ [Sentence α is **f** in interpretation i]

Semantic Rules

$\models_i$ true for all i
$\not\models_i$ false for all i [the sentence false has truth value **f** in all interpret.]

$\models_i \neg\alpha$ if and only if $\not\models_i \alpha$
$\models_i \alpha \wedge \beta$ if and only if $\models_i \alpha$ and $\models_i \beta$ [conjunction]
$\models_i \alpha \vee \beta$ if and only if $\models_i \alpha$ or $\models_i \beta$ [disjunction]
$\models_i P$ iff $i(P) = \mathbf{t}$

Models

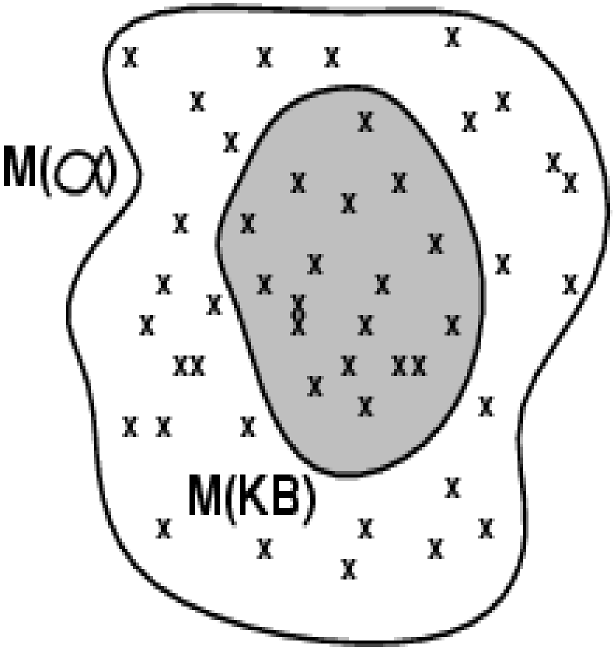
Logicians typically think in terms of models

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 $\models$ α iff $M(KB) \subseteq M(\alpha)$

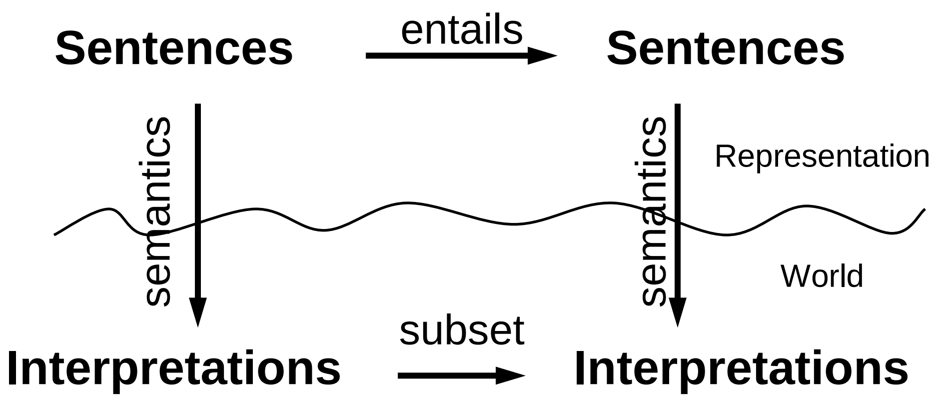
E.g. *KB* = Giants won and Reds won
α = Giants won



Models and Entailment

An interpretation i is a model of a sentence α iff $\models_i \alpha$

A set of sentences KB entails α iff every model of KB is also a model of α



Deduction Theorem:

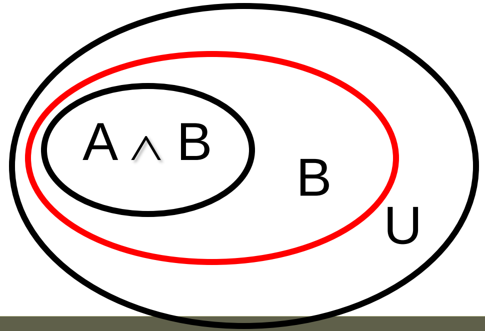
$$KB \models \alpha \text{ iff } \models KB \Rightarrow \alpha$$

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

$KB = A \wedge B$

$\alpha = B$

$A \wedge B \models B$

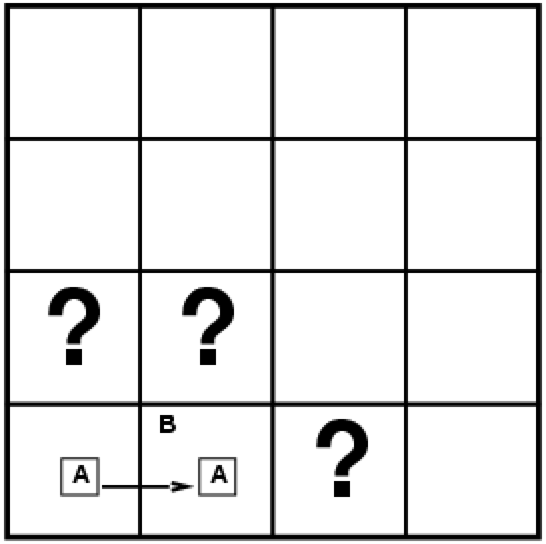


Entailment in the wumpus world

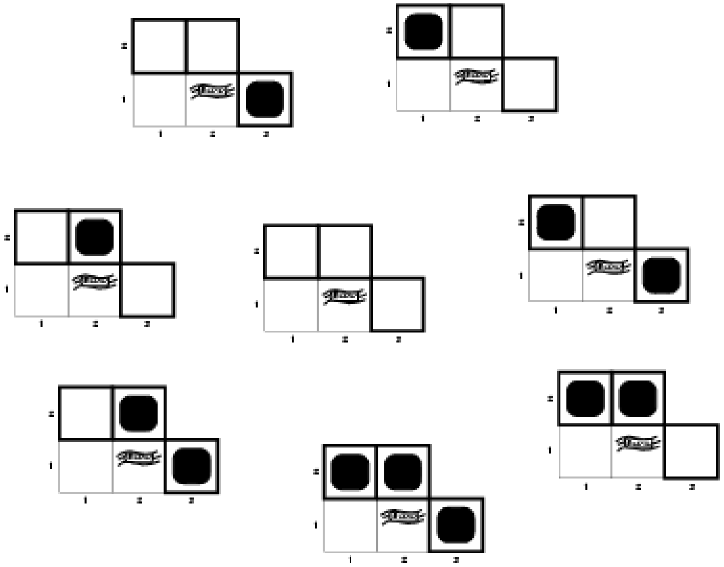
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

Consider possible models for *KB* assuming only pits

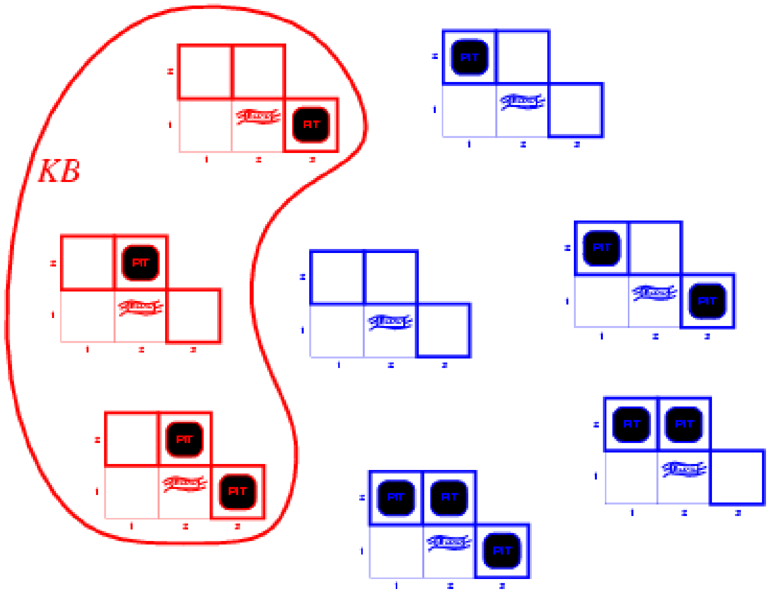
3 Boolean choices $\Rightarrow$ 8 possible models



Wumpus models

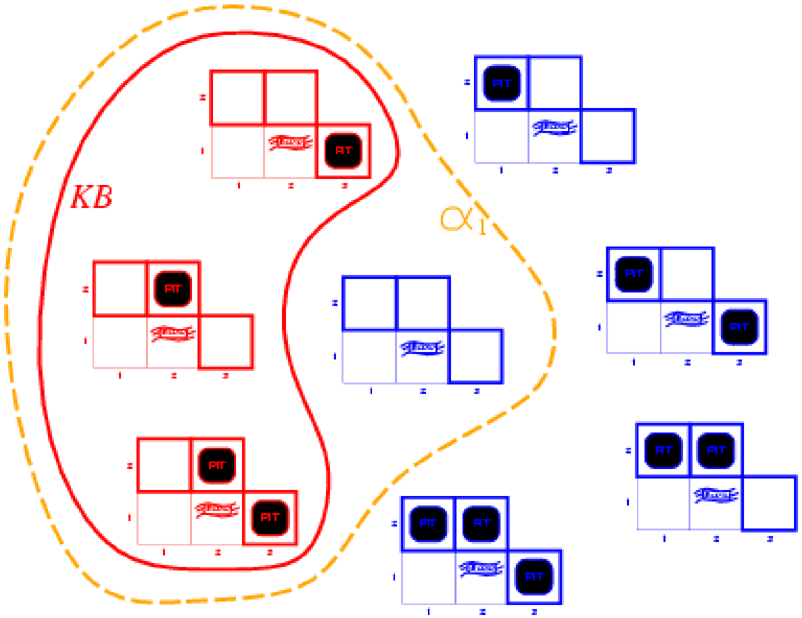


Wumpus models



KB = wumpus-world rules + observations

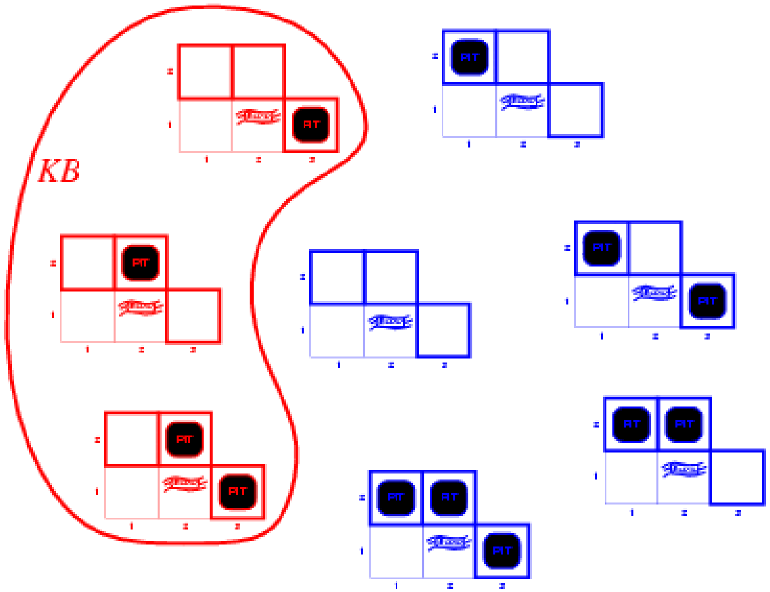
Wumpus models



KB = wumpus-world rules + observations

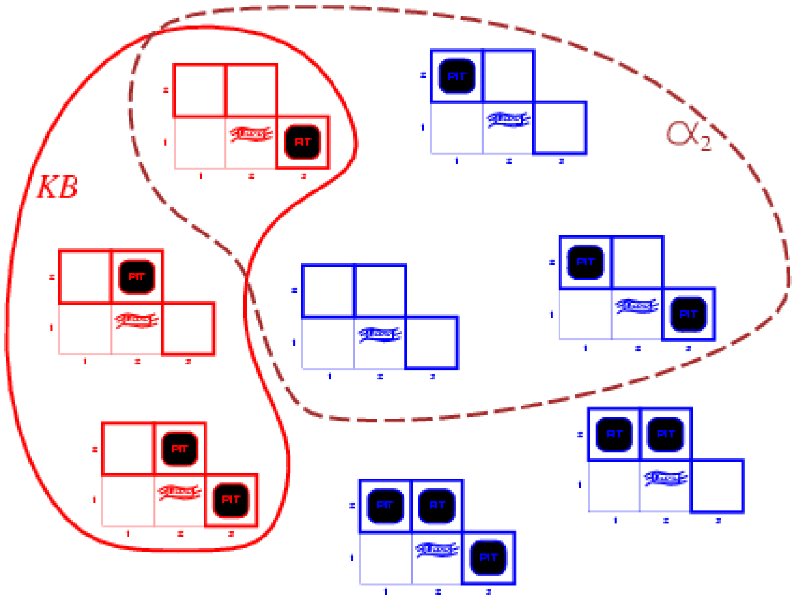
α_1 = "[1,2] is safe", *KB* $\models \alpha_1$, proved by model checking

Wumpus models



KB = wumpus-world rules + observations

Wumpus models



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

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].

Let KB include the following 5 rules:

R1: $\neg P_{1,1}$
R2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

R4: $\neg B_{1,1}$
R5: $B_{2,1}$
and α_1 = "[1,2] is safe"

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))
```

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*, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

Validity is connected to the inference via:

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

A sentence is satisfiable if it is true in some model

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

A sentence is unsatisfiable if it is true in no models

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

Satisfiability is connected to the inference via:

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

Rules of Inference

$P \wedge Q$ ——— $\therefore P$	And Elimination (- $\wedge$)	$P \vee Q$ $\neg P$ ——— $\therefore Q$	Or Elimination (- $\vee$)
P Q ——— $\therefore P \wedge Q$	And Introduction (+ $\wedge$)	P ——— $\therefore P \vee Q$	Or Introduction (+ $\vee$)

Rules of Inference

$$\begin{array}{l} \neg\neg P \\ \hline \therefore P \end{array}$$

Double Negaton
(- ¬)

$$\begin{array}{l} P \Rightarrow Q \\ P \\ \hline \therefore Q \end{array}$$

Modus Ponens
(- ⇒)

$$\begin{array}{|l} \underline{P} \\ \\ Q \\ \neg Q \end{array}$$
$\therefore \neg P$

Reductio Ad Absurdum
(+ ¬)

$$\begin{array}{|l} \underline{P} \\ \\ Q \end{array}$$
$\therefore P \Rightarrow Q$

Conditional Proof
(+ ⇒)

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., hill-climbing like algorithms

Resolution

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literals

clauses

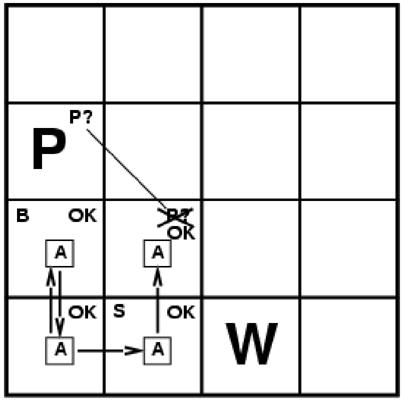
E.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

Resolution inference rule (for CNF):

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

where l_i and m_j are complementary literals.

E.g., $\dfrac{P_{1,3} \lor P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$



Resolution is sound and complete for propositional logic

Conversion to CNF

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

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

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

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

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

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

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

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

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

Resolution algorithm

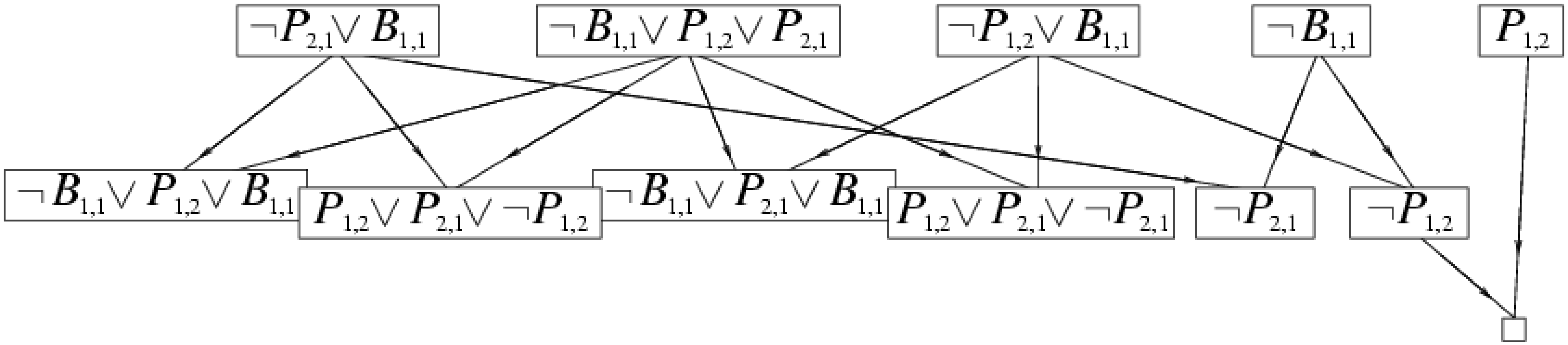
Proof by contradiction, i.e., show *KB*∧¬α unsatisfiable

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each Ci, Cj in clauses do
            resolvents ← PL-RESOLVE(Ci, Cj)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Resolution example

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

$\alpha = \neg P_{1,2}$



Propositional Resolution - An Example

P	P	(1)
$(P \wedge Q) \Rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \Rightarrow Q$	$\neg S \vee Q$	(3)
	$\neg T \vee Q$	(4)
T	T	(5)

Prove R:

$\neg P \vee \neg Q \vee R$ $\qquad$ $\neg R$

$\neg P \vee \neg Q$ $\qquad$ P

$\neg T \vee Q$ $\qquad$ $\neg Q$

$\neg T$ $\qquad$ T

nil

Propositional Resolution – Only Select One Pair to Resolve

$P \vee Q$ (1)

$\neg P \vee \neg Q \vee R$ (2)

Prove R:

$\neg P \vee \neg Q \vee R$ $\neg R$

$\neg P \vee \neg Q$ $P \vee Q$

nil

But is R entailed by the two facts we have been given?

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

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

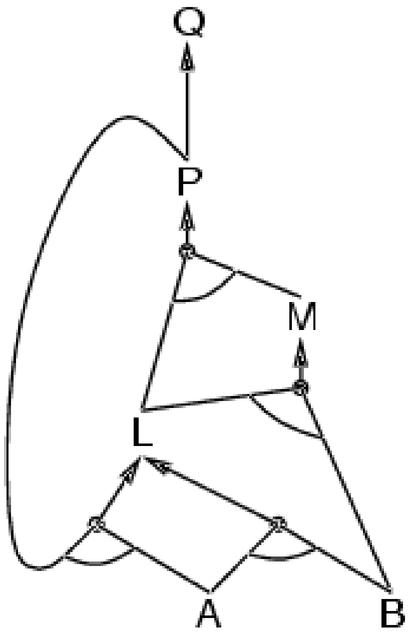
Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in linear time

Forward chaining

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

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

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

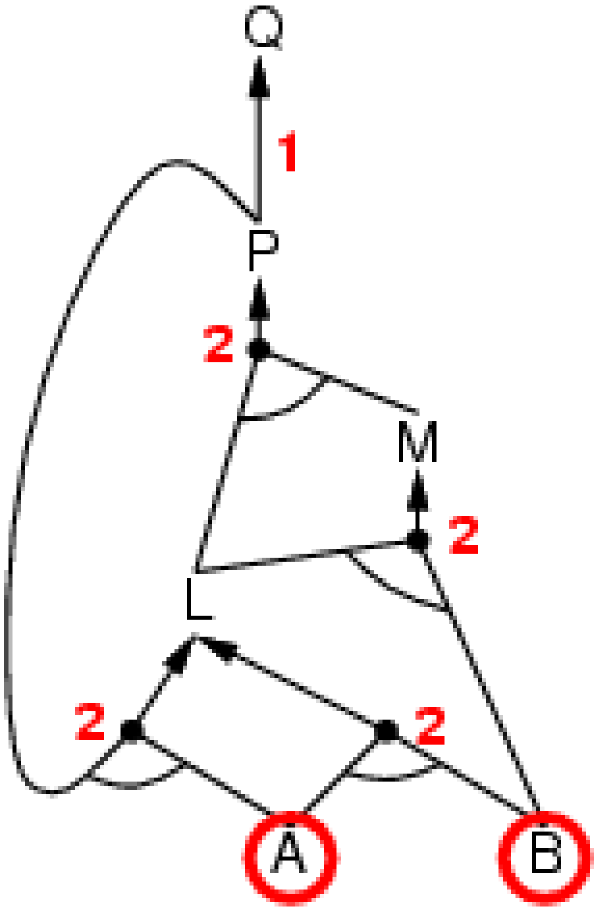


Forward chaining algorithm

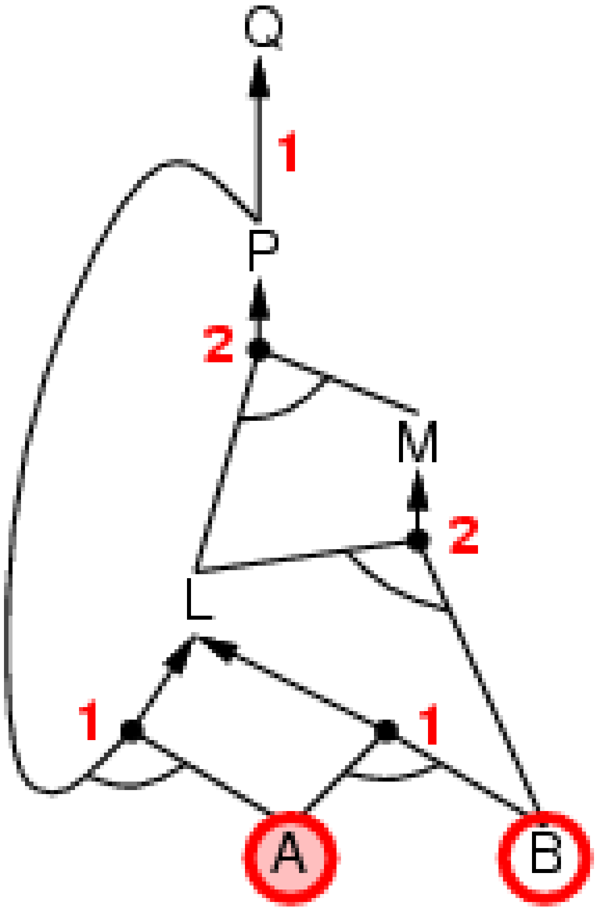
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

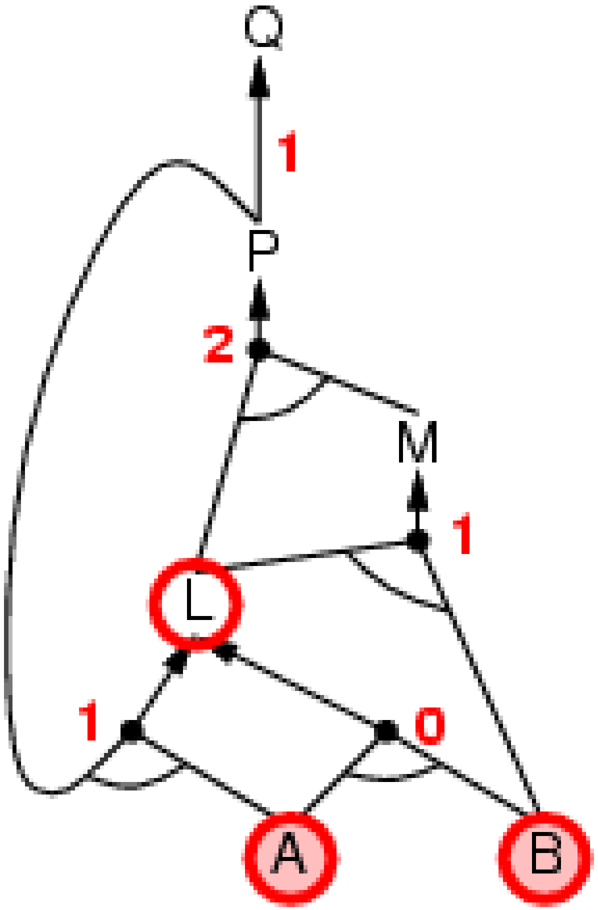
Forward chaining example



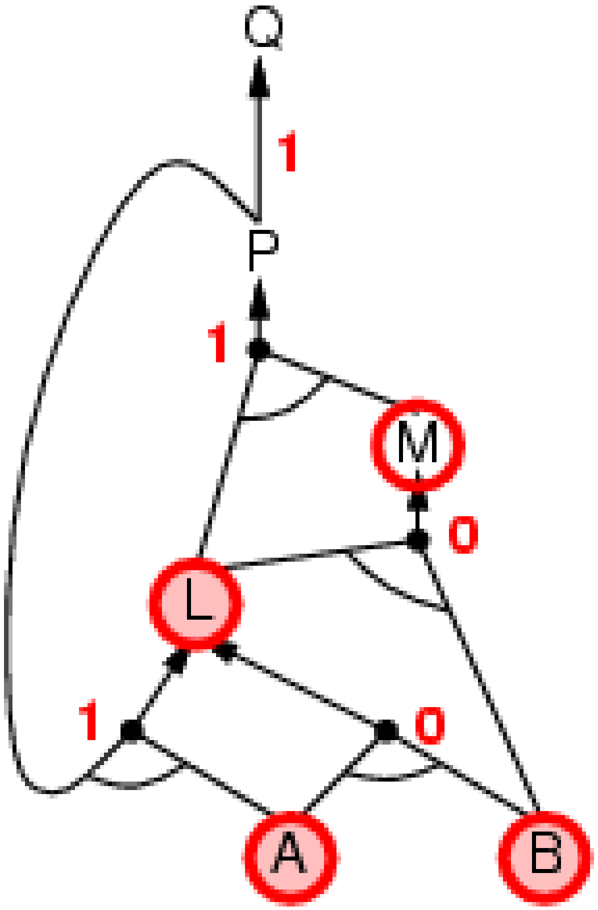
Forward chaining example



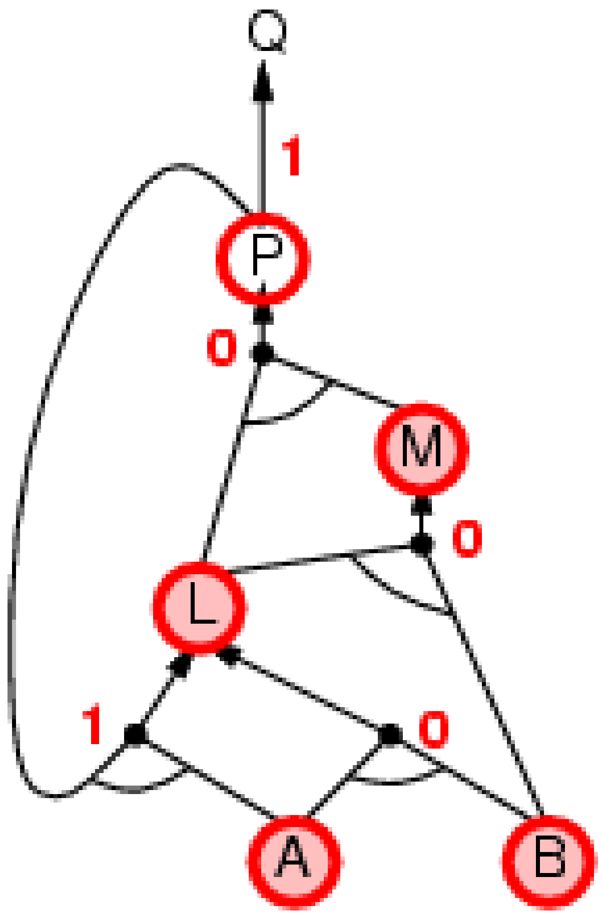
Forward chaining example



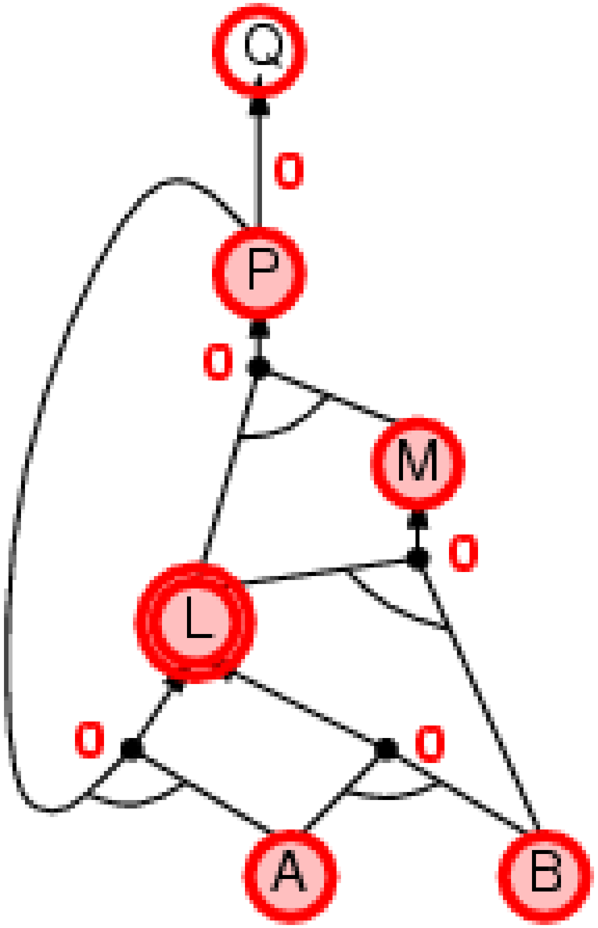
Forward chaining example



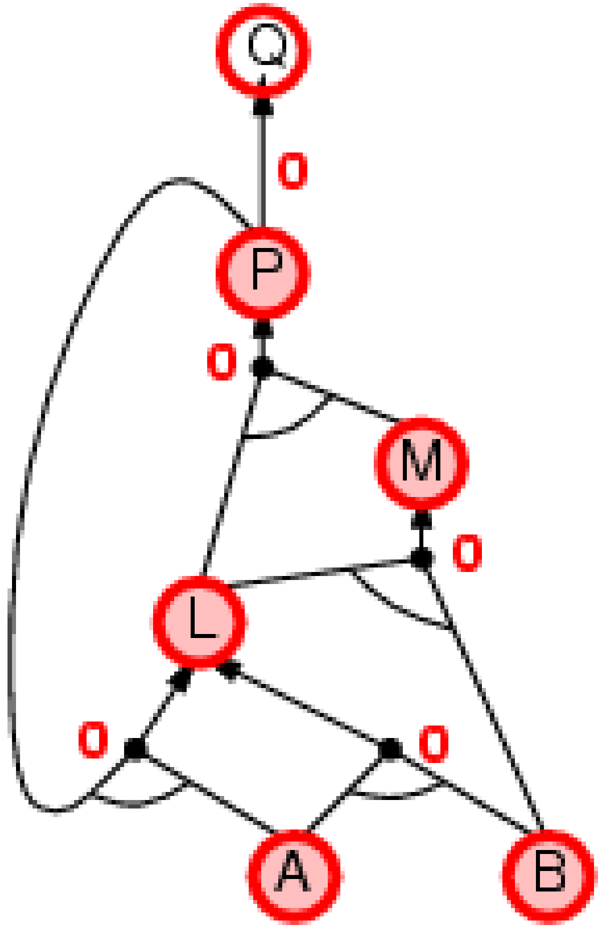
Forward chaining example



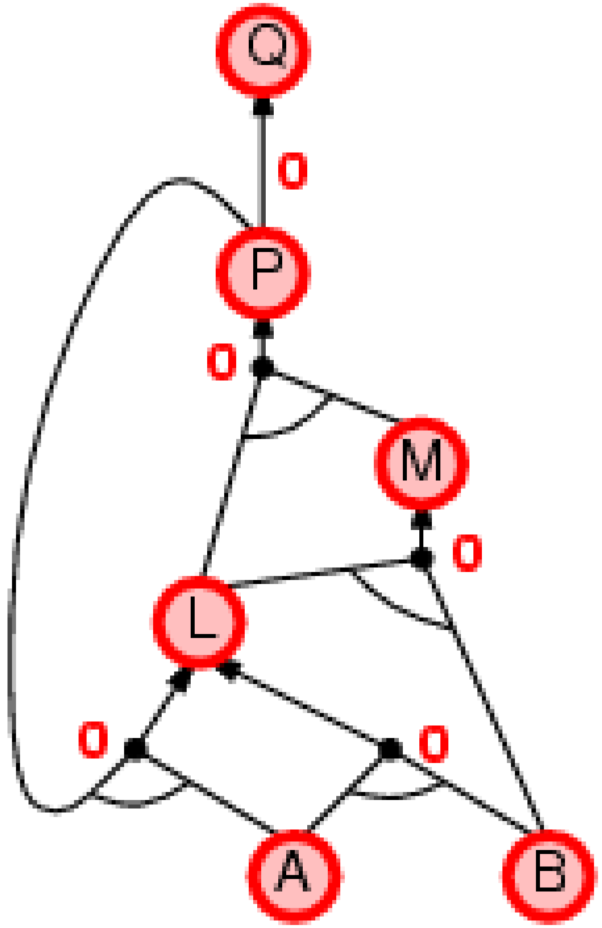
Forward chaining example



Forward chaining example



Forward chaining example



Backward chaining

Idea: work backwards from the query *q*:

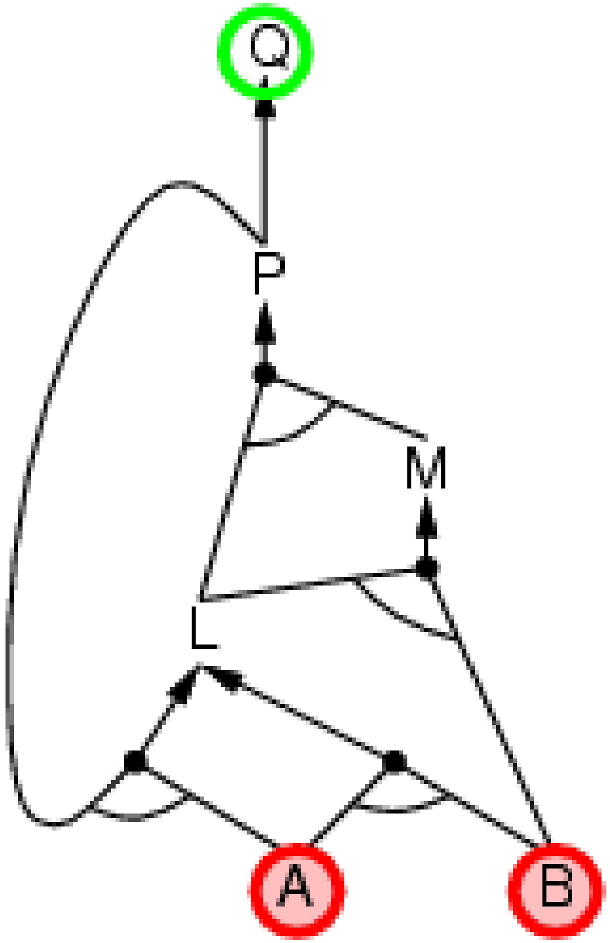
- to prove *q* by BC,
  - check if *q* is known already, or
  - prove by BC all premises of some rule concluding *q*

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

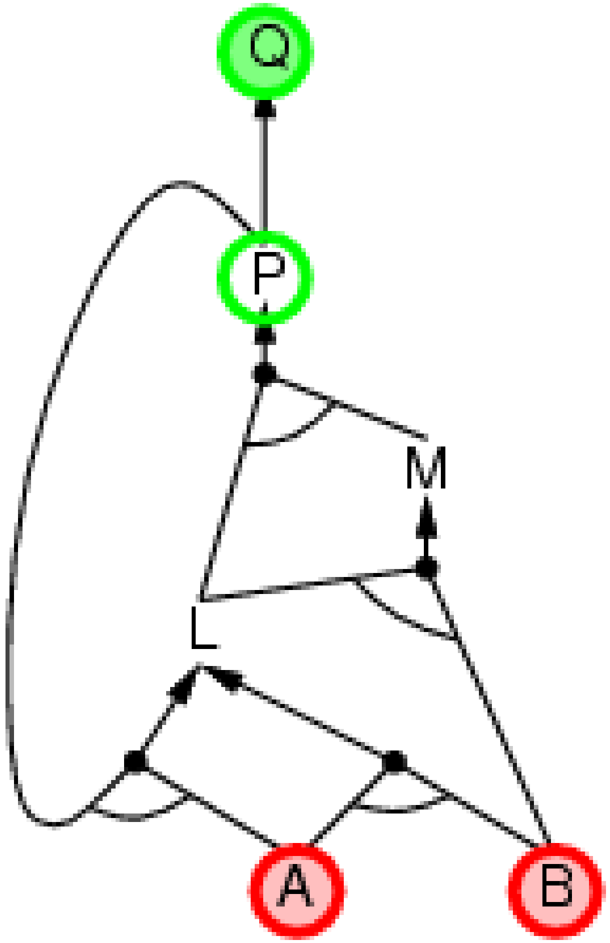
Avoid repeated work: check if new subgoal

- has already been proved true, or
- has already failed

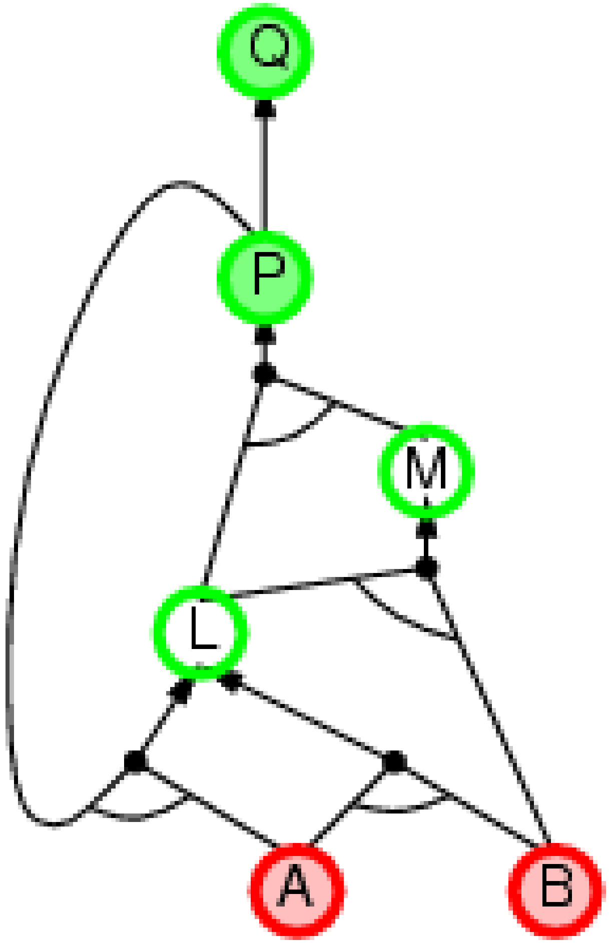
Backward chaining example



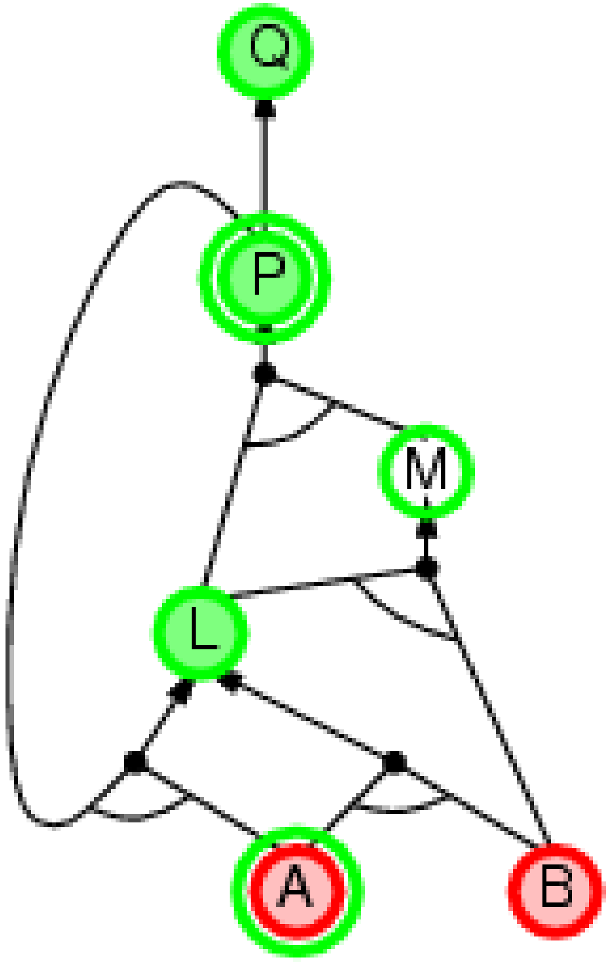
Backward chaining example



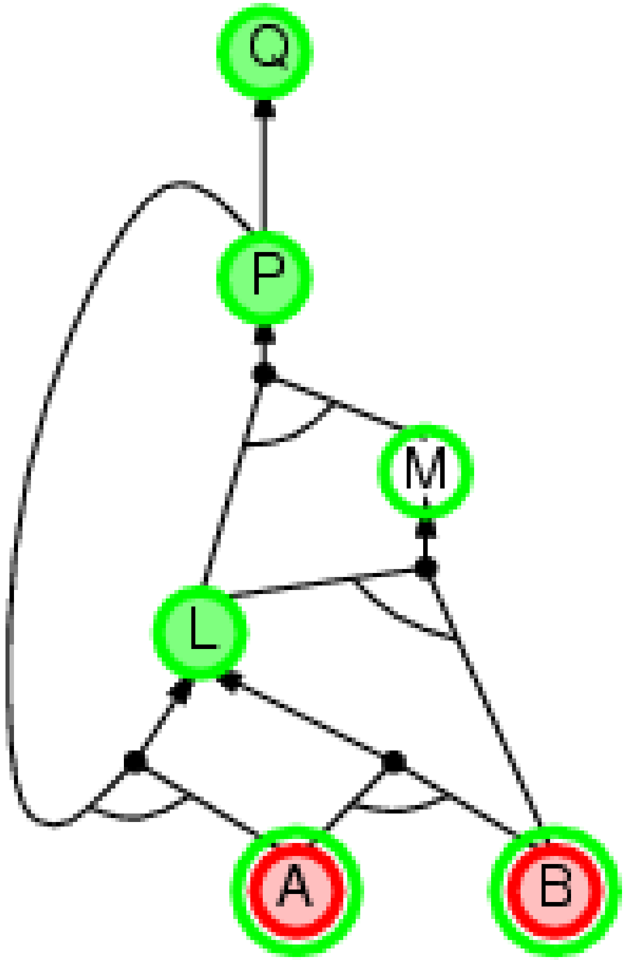
Backward chaining example



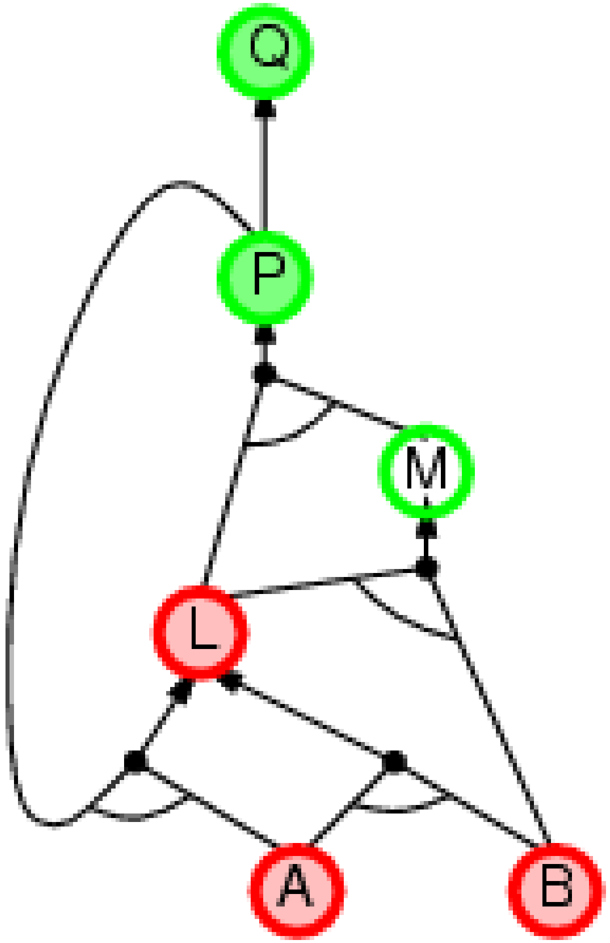
Backward chaining example



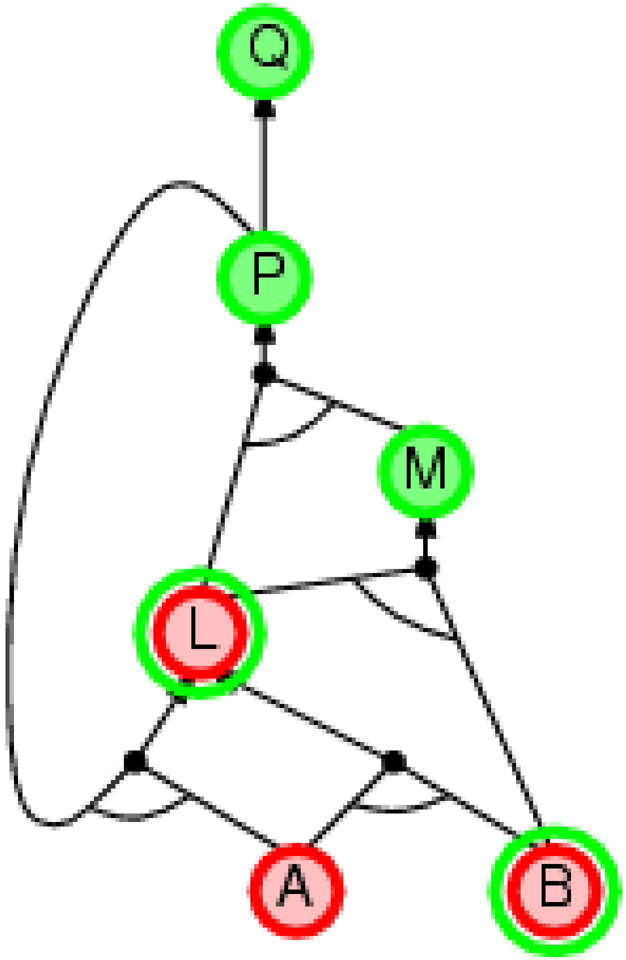
Backward chaining example



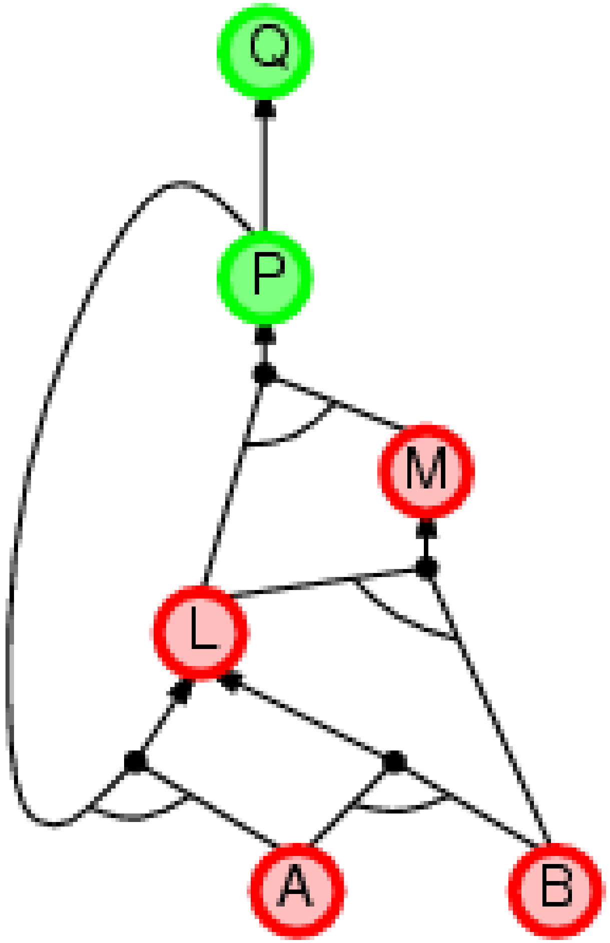
Backward chaining example



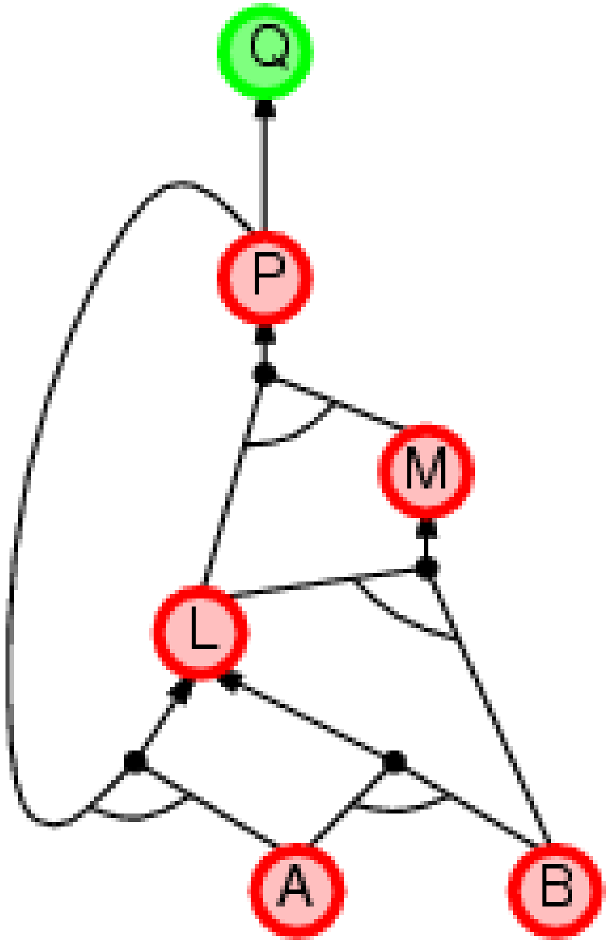
Backward chaining example



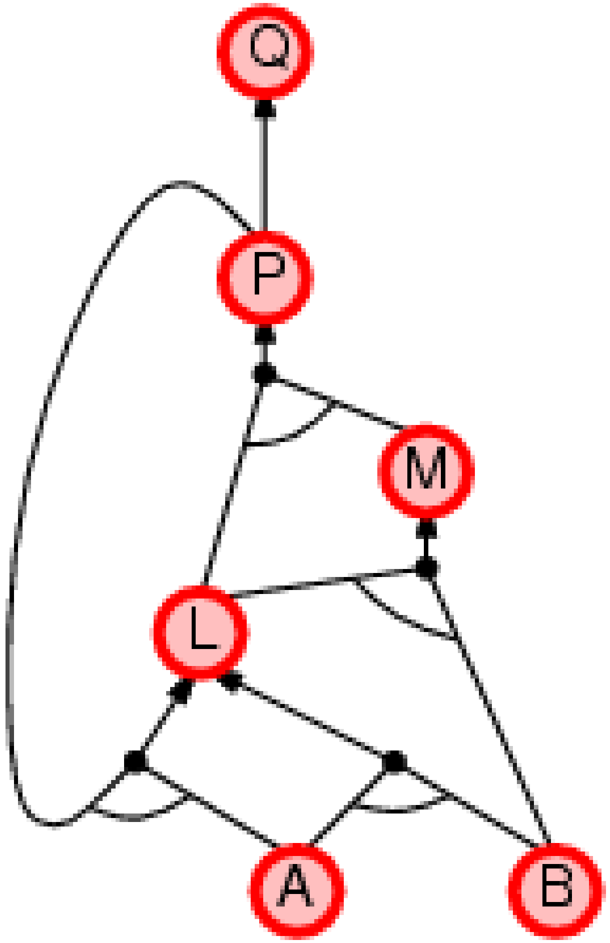
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

FC is data-driven, automatic, unconscious processing

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving

Complexity of BC can be much less than linear in the 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
```